

Teoria Formale delle Lingue Naturali

Parte I: Fondamenti Matematici e Teoria degli Automi

Capitolo 1: Strutture Algebriche Preliminari

1.1 Semigrupp, Monoidi e la Struttura delle Stringhe

La teoria formale delle lingue naturali richiede innanzitutto la definizione rigorosa degli oggetti matematici fondamentali che costituiscono il substrato su cui operano le grammatiche formali. Iniziamo con la nozione di alfabeto e le strutture algebriche ad esso associate.

Sia Σ un insieme finito non vuoto, che denominiamo alfabeto. Gli elementi di Σ sono detti simboli o lettere. Una stringa o parola su Σ è una sequenza finita di elementi di Σ . Formalmente, definiamo l'insieme di tutte le stringhe di lunghezza n su Σ come Σ^n , dove Σ^0 contiene unicamente la stringa vuota, denotata con ϵ .

Definizione 1.1.1 (Chiusura di Kleene): La chiusura di Kleene di un alfabeto Σ , denotata Σ^* , è definita come l'unione di tutti gli insiemi di stringhe di lunghezza finita:

$$\Sigma^* = \bigcup_{n=0 \text{ fino a } \infty} \Sigma^n$$

La chiusura positiva di Kleene, denotata Σ^+ , esclude la stringa vuota:

$$\Sigma^+ = \bigcup_{n=1 \text{ fino a } \infty} \Sigma^n = \Sigma^* \setminus \{\epsilon\}$$

Definizione 1.1.2 (Operazione di concatenazione): Siano $w_1 \in \Sigma^*$ e $w_2 \in \Sigma^*$ due stringhe su Σ . La concatenazione di w_1 e w_2 , denotata w_1w_2 o $w_1 \cdot w_2$, è la stringa ottenuta giustapponendo i simboli di w_2 dopo quelli di w_1 . Formalmente, se $w_1 = a_1a_2\dots a_n$ e $w_2 = b_1b_2\dots b_m$, allora $w_1w_2 = a_1a_2\dots a_nb_1b_2\dots b_m$.

Teorema 1.1.1 (Struttura monoide di Σ^*): La coppia $(\Sigma^*, \cdot)$ forma un monoide libero con elemento neutro ϵ .

Dimostrazione: Dobbiamo verificare tre proprietà fondamentali.

(i) Chiusura: Per ogni $w_1, w_2 \in \Sigma^*$, la concatenazione w_1w_2 è ancora un elemento di Σ^* . Questo è immediato dalla definizione di concatenazione, poiché la giustapposizione di due sequenze finite produce una sequenza finita.

(ii) Associatività: Per ogni $w_1, w_2, w_3 \in \Sigma^*$, vale $(w_1w_2)w_3 = w_1(w_2w_3)$. Sia $w_1 = a_1\dots a_n$, $w_2 = b_1\dots b_m$, $w_3 = c_1\dots c_k$. Allora: $(w_1w_2)w_3 = (a_1\dots a_nb_1\dots b_m)c_1\dots c_k = a_1\dots a_nb_1\dots b_mc_1\dots c_k = w_1(w_2w_3) = a_1\dots a_n(b_1\dots b_mc_1\dots c_k) = a_1\dots a_nb_1\dots b_mc_1\dots c_k$. Dunque le due espressioni coincidono.

(iii) Esistenza dell'elemento neutro: Per ogni $w \in \Sigma^*$, vale $\varepsilon w = w\varepsilon = w$. Se $w = a_1 \dots a_n$, allora εw è la concatenazione della stringa vuota con w , che per definizione produce w . Analogamente $w\varepsilon = w$.

La libertà del monoide deriva dal fatto che ogni elemento di Σ^* ha una rappresentazione unica come prodotto di elementi di Σ (interpretati come stringhe di lunghezza unitaria), e non esistono relazioni non banali tra gli elementi dell'alfabeto. ■

Definizione 1.1.3 (Lunghezza di una stringa): La funzione lunghezza $|\cdot| : \Sigma^* \rightarrow \mathbb{N}$ è definita ricorsivamente da:

- $|\varepsilon| = 0$
- $|wa| = |w| + 1$, per ogni $w \in \Sigma^*$ e $a \in \Sigma$

Lemma 1.1.2: Per ogni $w_1, w_2 \in \Sigma^*$, vale $|w_1 w_2| = |w_1| + |w_2|$.

Dimostrazione: Procediamo per induzione sulla lunghezza di w_2 .

Base: Se $|w_2| = 0$, allora $w_2 = \varepsilon$ e $|w_1 w_2| = |w_1 \varepsilon| = |w_1| = |w_1| + 0 = |w_1| + |\varepsilon|$.

Passo induttivo: Supponiamo che la proprietà valga per tutte le stringhe di lunghezza k , e sia w_2 una stringa di lunghezza $k+1$. Allora $w_2 = w'a$ per qualche w' con $|w'| = k$ e $a \in \Sigma$. Abbiamo: $|w_1 w_2| = |w_1 (w'a)| = |(w_1 w')a| = |w_1 w'| + 1$ Per ipotesi induttiva, $|w_1 w'| = |w_1| + |w'|$, quindi: $|w_1 w_2| = |w_1| + |w'| + 1 = |w_1| + (|w'| + 1) = |w_1| + |w_2|$. ■

Definizione 1.1.4 (Potenza di una stringa): Per $w \in \Sigma^*$ e $n \in \mathbb{N}$, definiamo ricorsivamente:

- $w^0 = \varepsilon$
- $w^{n+1} = w^n \cdot w$

Definizione 1.1.5 (Inversione di stringa): La funzione di inversione $(\cdot)^R : \Sigma^* \rightarrow \Sigma^*$ è definita ricorsivamente da:

- $\varepsilon^R = \varepsilon$
- $(wa)^R = a(w^R)$, per ogni $w \in \Sigma^*$ e $a \in \Sigma$

Lemma 1.1.3: Per ogni $w_1, w_2 \in \Sigma^*$, vale $(w_1 w_2)^R = w_2^R w_1^R$.

Dimostrazione: Procediamo per induzione sulla lunghezza di w_2 .

Base: Se $w_2 = \varepsilon$, allora $(w_1 \varepsilon)^R = w_1^R = \varepsilon w_1^R = \varepsilon^R w_1^R$.

Passo induttivo: Sia $w_2 = w'a$ con $a \in \Sigma$. Allora: $(w_1 w_2)^R = (w_1 (w'a))^R = ((w_1 w')a)^R = a((w_1 w')^R)$ Per ipotesi induttiva, $(w_1 w')^R = w'^R w_1^R$, quindi: $(w_1 w_2)^R = a(w'^R w_1^R) = (aw'^R)w_1^R = (w'a)^R w_1^R = w_2^R w_1^R$. ■

1.2 Linguaggi Formali e Operazioni

Definizione 1.2.1 (Linguaggio formale): Un linguaggio formale L su un alfabeto Σ è un sottoinsieme di Σ^* , ovvero $L \subseteq \Sigma^*$. L'insieme di tutti i linguaggi su Σ è dunque l'insieme delle parti di Σ^* , denotato $\mathcal{P}(\Sigma^*)$.

Definizione 1.2.2 (Operazioni sui linguaggi): Siano $L_1, L_2 \subseteq \Sigma^*$ due linguaggi. Definiamo:

- (i) Unione: $L_1 \cup L_2 = \{w \in \Sigma^* : w \in L_1 \text{ oppure } w \in L_2\}$
- (ii) Intersezione: $L_1 \cap L_2 = \{w \in \Sigma^* : w \in L_1 \text{ e } w \in L_2\}$
- (iii) Complemento: $\bar{L} = \Sigma^* \setminus L = \{w \in \Sigma^* : w \notin L\}$
- (iv) Concatenazione: $L_1 L_2 = \{w_1 w_2 : w_1 \in L_1 \text{ e } w_2 \in L_2\}$
- (v) Chiusura di Kleene: $L^* = \bigcup_{n=0}^{\infty} L^n$, dove: $L^0 = \{\epsilon\}$ - $L^{n+1} = L L^n$
- (vi) Chiusura positiva: $L^+ = \bigcup_{n=1}^{\infty} L^n = L L^*$

Teorema 1.2.1: L'insieme $(\mathcal{P}(\Sigma^*), \cup, \cap, \bar{}, \emptyset, \Sigma^*)$ forma un'algebra booleana.

Dimostrazione: Verifichiamo gli assiomi dell'algebra booleana.

- (i) Commutatività: $L_1 \cup L_2 = L_2 \cup L_1$ e $L_1 \cap L_2 = L_2 \cap L_1$ seguono direttamente dalle proprietà dell'unione e intersezione insiemistiche.
- (ii) Associatività: $(L_1 \cup L_2) \cup L_3 = L_1 \cup (L_2 \cup L_3)$ e $(L_1 \cap L_2) \cap L_3 = L_1 \cap (L_2 \cap L_3)$ seguono dalle proprietà insiemistiche.
- (iii) Distributività: $L_1 \cup (L_2 \cap L_3) = (L_1 \cup L_2) \cap (L_1 \cup L_3)$ e $L_1 \cap (L_2 \cup L_3) = (L_1 \cap L_2) \cup (L_1 \cap L_3)$ sono le leggi distributive della teoria degli insiemi.
- (iv) Elementi identità: $\emptyset$ è l'identità per l'unione ($L \cup \emptyset = L$) e Σ^* è l'identità per l'intersezione ($L \cap \Sigma^* = L$).
- (v) Complementazione: Per ogni $L \subseteq \Sigma^*$, vale $L \cup \bar{L} = \Sigma^*$ e $L \cap \bar{L} = \emptyset$, dove $\bar{L} = \Sigma^* \setminus L$.

Queste proprietà, insieme alla legge di De Morgan e agli altri assiomi, stabiliscono che $(\mathcal{P}(\Sigma^*), \cup, \cap, \bar{}, \emptyset, \Sigma^*)$ è un'algebra booleana completa. ■

Teorema 1.2.2 (Proprietà della concatenazione di linguaggi): La concatenazione di linguaggi soddisfa le seguenti proprietà:

- (i) Associatività: $(L_1 L_2) L_3 = L_1 (L_2 L_3)$ (ii) Distributività rispetto all'unione: $L_1 (L_2 \cup L_3) = L_1 L_2 \cup L_1 L_3$ e $(L_1 \cup L_2) L_3 = L_1 L_3 \cup L_2 L_3$ (iii) Elemento neutro: $\{\epsilon\} L = L \{\epsilon\} = L$ (iv) Elemento assorbente: $\emptyset L = L \emptyset = \emptyset$

Dimostrazione:

- (i) Sia $w \in (L_1 L_2) L_3$. Allora esistono $u \in L_1 L_2$ e $v \in L_3$ tali che $w = uv$. Poiché $u \in L_1 L_2$, esistono $u_1 \in L_1$ e $u_2 \in L_2$ tali che $u = u_1 u_2$. Quindi $w = u_1 u_2 v = u_1 (u_2 v)$. Poiché $u_2 v \in L_2 L_3$, abbiamo $w \in L_1 (L_2 L_3)$. L'inclusione opposta si dimostra analogamente.
- (ii) Sia $w \in L_1 (L_2 \cup L_3)$. Allora esistono $u \in L_1$ e $v \in L_2 \cup L_3$ tali che $w = uv$. Se $v \in L_2$, allora $w \in L_1 L_2$; se $v \in L_3$, allora $w \in L_1 L_3$. In entrambi i casi, $w \in L_1 L_2 \cup L_1 L_3$. Viceversa,

se $w \in L_1 L_2 \cup L_1 L_3$, allora $w \in L_1 L_2$ oppure $w \in L_1 L_3$, quindi esistono $u \in L_1$ e $v \in L_2$ oppure $v \in L_3$, da cui $v \in L_2 \cup L_3$ e $w \in L_1(L_2 \cup L_3)$.

(iii) Se $w \in \{\varepsilon\}L$, allora $w = \varepsilon v$ per qualche $v \in L$, quindi $w = v \in L$. Viceversa, se $w \in L$, allora $w = \varepsilon w \in \{\varepsilon\}L$.

(iv) Se $w \in \emptyset L$, allora esistono $u \in \emptyset$ e $v \in L$ tali che $w = uv$, il che è impossibile poiché $\emptyset$ non contiene elementi. Quindi $\emptyset L = \emptyset$. ■

Teorema 1.2.3 (Equazione di punto fisso di Arden): Sia $L \subseteq \Sigma^*$ un linguaggio tale che $\varepsilon \notin L$. Allora l'equazione $X = LX \cup M$ ha come unica soluzione $X = L^*M$.

Dimostrazione: Verifichiamo prima che LM è una soluzione: $L(LM) \cup M = L(\cup_{n=0}^{\infty} L^n M) \cup M = \cup_{n=0}^{\infty} L^{n+1} M \cup M = \cup_{n=0}^{\infty} L^{n+1} M \cup L^0 M = \cup_{n=0}^{\infty} L^{n+1} M \cup L^0 M = L^*M$

Per l'unicità, supponiamo che X sia una soluzione. Allora $X = LX \cup M$. Sostituendo iterativamente: $X = LX \cup M = L(LX \cup M) \cup M = L^2X \cup LM \cup M$ Continuando: $X = L^n X \cup L^{n-1}M \cup \dots \cup LM \cup M$

Affermiamo che $X \subseteq LM$. Sia $w \in X$ con $|w| = k$. Scegliendo n sufficientemente grande ($n > k$), ogni stringa in $L^n X$ ha lunghezza maggiore di k (poiché $\varepsilon \notin L$, ogni stringa in L ha lunghezza almeno 1). Quindi $w \notin L^n X$, e deve appartenere a $L^{n-1}M \cup \dots \cup LM \cup M = (L^{n-1} \cup \dots \cup L \cup \{\varepsilon\})M \subseteq LM$.

D'altra parte, da $X = LX \cup M$ segue $M \subseteq X$. Inoltre, se $X \supseteq L^n M$ per qualche n , allora: $X \supseteq LX \supseteq L(L^n M) = L^{n+1}M$ Per induzione, $X \supseteq L^n M$ per ogni $n \geq 0$, quindi $X \supseteq LM$. Combinando le due inclusioni, $X = LM$. ■

1.3 Relazioni, Congruenze e Quozienti

Definizione 1.3.1 (Relazione di equivalenza su Σ^*): Una relazione binaria $\equiv \subseteq \Sigma^* \times \Sigma^*$ è una relazione di equivalenza se soddisfa: (i) Riflessività: $w \equiv w$ per ogni $w \in \Sigma^*$ (ii) Simmetria: se $w_1 \equiv w_2$, allora $w_2 \equiv w_1$ (iii) Transitività: se $w_1 \equiv w_2$ e $w_2 \equiv w_3$, allora $w_1 \equiv w_3$

Definizione 1.3.2 (Congruenza destra): Una relazione di equivalenza $\equiv$ su Σ^* è una congruenza destra se per ogni $w_1, w_2, u \in \Sigma^*$: $w_1 \equiv w_2$ implica $w_1 u \equiv w_2 u$

Definizione 1.3.3 (Congruenza sinistra): Una relazione di equivalenza $\equiv$ su Σ^* è una congruenza sinistra se per ogni $w_1, w_2, u \in \Sigma^*$: $w_1 \equiv w_2$ implica $u w_1 \equiv u w_2$

Definizione 1.3.4 (Congruenza): Una relazione di equivalenza è una congruenza se è sia una congruenza destra che una congruenza sinistra.

Definizione 1.3.5 (Indice di una relazione di equivalenza): L'indice di una relazione di equivalenza $\equiv$ è il numero di classi di equivalenza distinte indotte da $\equiv$ su Σ^* .

Definizione 1.3.6 (Congruenza di Myhill-Nerode): Sia $L \subseteq \Sigma^*$ un linguaggio. La congruenza di Myhill-Nerode $\equiv_L$ è definita da: $x \equiv_L y$ se e solo se per ogni $z \in \Sigma^*$, $xz \in L \Leftrightarrow yz \in L$

Teorema 1.3.1: La relazione $\equiv_L$ è una congruenza destra di indice finito se e solo se L è riconoscibile da un automa a stati finiti.

Dimostrazione: ($\Rightarrow$) Supponiamo che $\equiv_L$ abbia indice finito n . Sia $Q = \{[w]_L : w \in \Sigma^*\}$ l'insieme delle classi di equivalenza, con $|Q| = n$. Definiamo un automa a stati finiti deterministico $A = (Q, \Sigma, \delta, q_0, F)$ dove:

- Q è l'insieme delle classi di equivalenza
- $\delta([w], a) = [wa]$ per ogni $[w] \in Q$ e $a \in \Sigma$
- $q_0 = [\varepsilon]$
- $F = \{[w] : w \in L\}$

La funzione δ è ben definita perché $\equiv_L$ è una congruenza destra: se $w_1 \equiv_L w_2$, allora $w_1 a \equiv_L w_2 a$ per ogni $a \in \Sigma$, quindi $[w_1 a] = [w_2 a]$.

Estendiamo δ a $\delta^* : Q \times \Sigma^* \rightarrow Q$ definendo:

- $\delta^*([w], \varepsilon) = [w]$
- $\delta^*([w], ua) = \delta(\delta^*([w], u), a) = \delta([wu], a) = [wua]$

Per induzione su $|u|$, si mostra che $\delta^*([\varepsilon], u) = [u]$. Quindi A accetta una stringa w se e solo se $\delta^*([\varepsilon], w) = [w] \in F$, cioè se e solo se $w \in L$.

($\Leftarrow$) Supponiamo che L sia riconosciuto da un automa $A = (Q, \Sigma, \delta, q_0, F)$ con $|Q| = n$. Definiamo $w_1 \approx_A w_2$ se $\delta^*(q_0, w_1) = \delta^*(q_0, w_2)$. Chiaramente $\approx_A$ ha al massimo n classi di equivalenza. Mostriamo che $w_1 \approx_A w_2$ implica $w_1 \equiv_L w_2$.

Sia $z \in \Sigma^*$ arbitrario. Se $\delta^*(q_0, w_1) = \delta^*(q_0, w_2) = q$, allora: $\delta^*(q_0, w_1 z) = \delta^*(\delta^*(q_0, w_1), z) = \delta^*(q, z)$ $\delta^*(q_0, w_2 z) = \delta^*(\delta^*(q_0, w_2), z) = \delta^*(q, z)$

Quindi $w_1 z \in L$ se e solo se $\delta^*(q_0, w_1 z) \in F$, che equivale a $\delta^*(q, z) \in F$, che a sua volta equivale a $\delta^*(q_0, w_2 z) \in F$, cioè $w_2 z \in L$. Dunque $w_1 \equiv_L w_2$.

Poiché $\approx_A$ raffina $\equiv_L$ (nel senso che $w_1 \approx_A w_2$ implica $w_1 \equiv_L w_2$) e $\approx_A$ ha al massimo n classi, anche $\equiv_L$ ha indice al massimo n , quindi finito. ■

Capitolo 1 Espanso: Teoria Algebrica Avanzata delle Stringhe

1.1 Teoria dei Monoidi Liberi: Approfondimenti

1.1.1 Proprietà Universali dei Monoidi Liberi

*Definizione 1.1.1 (Proprietà universale del monoide libero):** Sia A un insieme. Il monoide libero A^* su A è caratterizzato dalla seguente proprietà universale: per ogni monoide M e ogni funzione $f: A \rightarrow M$, esiste un unico omomorfismo di monoidi $\varphi: A^* \rightarrow M$ tale che il seguente diagramma commuta:

$$\begin{array}{ccc}
 A & \xrightarrow{i} & A^* \\
 | & & | \\
 f & & \varphi \\
 | & & | \\
 \downarrow & & \downarrow \\
 M & \xleftarrow{\quad} & M
 \end{array}$$

dove $i: A \rightarrow A^*$ è l'inclusione naturale $a \mapsto a$ (stringa di lunghezza 1).

Teorema 1.1.1 (Esistenza e unicità del monoide libero):* Per ogni insieme A , il monoide libero A^* esiste ed è unico a meno di isomorfismo.

Dimostrazione: *Esistenza:* Costruiamo esplicitamente A^* come l'insieme di tutte le sequenze finite di elementi di A , con concatenazione come operazione e la sequenza vuota ε come identità.

Unicità: Supponiamo che $(M_1, \cdot_1)$ e $(M_2, \cdot_2)$ soddisfino entrambi la proprietà universale. Allora esistono omomorfismi $\varphi_1: M_1 \rightarrow M_2$ e $\varphi_2: M_2 \rightarrow M_1$ tali che:

- $\varphi_2 \circ \varphi_1: M_1 \rightarrow M_1$ estende l'identità su A
- $\varphi_1 \circ \varphi_2: M_2 \rightarrow M_2$ estende l'identità su A

Per l'unicità della proprietà universale, $\varphi_2 \circ \varphi_1 = \text{id}_{\{M_1\}}$ e $\varphi_1 \circ \varphi_2 = \text{id}_{\{M_2\}}$, quindi φ_1 è un isomorfismo. ■

1.1.2 Funtorialità della Costruzione del Monoide Libero

Teorema 1.1.2 (Funtore monoide libero):* La costruzione $(-)^*: \text{Set} \rightarrow \text{Mon}$ è un funtore dalla categoria degli insiemi alla categoria dei monoidi.

Dimostrazione: Dobbiamo verificare che $(-)^*$ preservi identità e composizione.

(i) *Preservazione dell'identità:* Data una funzione identità $\text{id}_A: A \rightarrow A$, dobbiamo mostrare che $\text{id}_{A^*} = \text{id}_{\{A^*\}}$. Per ogni $w = a_1 a_2 \dots a_n \in A^*$:

$$\text{id}_{A^*}(w) = \text{id}_A(a_1) \cdot \text{id}_A(a_2) \cdot \dots \cdot \text{id}_A(a_n) = a_1 \cdot a_2 \cdot \dots \cdot a_n = w$$

(ii) *Preservazione della composizione:* Date $f: A \rightarrow B$ e $g: B \rightarrow C$, dobbiamo mostrare che $(g \circ f)^* = g^* \circ f^*$. Per ogni $w = a_1 \dots a_n \in A^*$:

$$(g \circ f)^*(w) = g(f(a_1)) \cdot \dots \cdot g(f(a_n))$$

$$= g^*(f(a_1) \cdot \dots \cdot f(a_n))$$

$$= g^*(f^*(w))$$

$$= (g^* \circ f^*)(w) \quad \blacksquare$$

1.1.3 Rappresentazione Matriciale dei Monoidi

Definizione 1.1.3 (Rappresentazione di monoidi):* Una rappresentazione di un monoide M in uno spazio vettoriale V è un omomorfismo $\rho: M \rightarrow \text{End}(V)$, dove $\text{End}(V)$ è il monoide degli endomorfismi lineari di V .

Teorema 1.1.3 (Teorema di rappresentazione di Cayley per monoidi):* Ogni monoide M può essere embedded in un monoide di trasformazioni.

Dimostrazione: Definiamo l'azione sinistra di M su se stesso: $\lambda: M \rightarrow (M \rightarrow M)$ da $\lambda(m)(x) = mx$.

Verifichiamo che λ è un omomorfismo iniettivo:

- $\lambda(e)(x) = ex = x$, quindi $\lambda(e) = \text{id}_M$
- $\lambda(m_1 m_2)(x) = (m_1 m_2)x = m_1(m_2 x) = \lambda(m_1)(\lambda(m_2)(x)) = (\lambda(m_1) \circ \lambda(m_2))(x)$

Quindi $\lambda(m_1 m_2) = \lambda(m_1) \circ \lambda(m_2)$.

Per l'iniettività: se $\lambda(m_1) = \lambda(m_2)$, allora per ogni $x \in M$, $m_1 x = m_2 x$. In particolare, prendendo $x = e$: $m_1 = m_1 e = m_2 e = m_2$. $\blacksquare$

1.1.4 Teorema del Sottomondo per Monoidi

Definizione 1.1.4 (Sottomonoide):* Un sottoinsieme S di un monoide M è un sottomonoide se:

1. $e \in S$ (contiene l'identità)
2. $\forall a, b \in S: ab \in S$ (chiuso sotto l'operazione)

Teorema 1.1.4 (Intersezione di sottomonoidi):* L'intersezione di una famiglia arbitraria di sottomonoidi di M è un sottomonoide di M .

Dimostrazione: Sia $\{S_i\}_{i \in I}$ una famiglia di sottomonoidi di M , e sia $S = \bigcap_{i \in I} S_i$.

(i) Poiché $e \in S_i$ per ogni $i \in I$, abbiamo $e \in S$.

(ii) Se $a, b \in S$, allora $a, b \in S_i$ per ogni $i \in I$. Poiché ogni S_i è chiuso sotto moltiplicazione, $ab \in S_i$ per ogni i , quindi $ab \in S$. $\blacksquare$

Definizione 1.1.5 (Sottomonoide generato):* Dato un sottoinsieme $X \subseteq M$, il sottomonoide generato da X , denotato $\langle X \rangle$, è l'intersezione di tutti i sottomonoidi contenenti X .

Teorema 1.1.5 (Forma esplicita del sottomonoide generato):*

$$\langle X \rangle = \{x_1 x_2 \dots x_n : n \geq 0, x_i \in X\} \cup \{e\}$$

Dimostrazione: Sia $S = \{x_1 x_2 \dots x_n : n \geq 0, x_i \in X\} \cup \{e\}$.

($\subseteq$) Chiaramente S è contenuto in ogni sottomonoido contenente X , quindi $S \subseteq \langle X \rangle$.

($\supseteq$) Verifichiamo che S è un sottomonoido:

- $e \in S$ per definizione
- Se $a = x_1 \dots x_n$ e $b = y_1 \dots y_m$ sono in S , allora $ab = x_1 \dots x_n y_1 \dots y_m \in S$

Quindi S è un sottomonoido contenente X , dunque $\langle X \rangle \subseteq S$. ■

1.2 Teoria Approfondita delle Congruenze

1.2.1 Congruenze e Quozienti

*Definizione 1.2.1 (Congruenza su monoidi):** Una relazione di equivalenza $\equiv$ su un monoido M è una congruenza se è compatibile con l'operazione:

$$\forall a, b, c, d \in M: (a \equiv b \wedge c \equiv d) \rightarrow ac \equiv bd$$

*Teorema 1.2.1 (Teorema di corrispondenza per monoidi):** Esiste una corrispondenza biunivoca tra:

1. Congruenze su M
2. Monoidi quoziente $M/\equiv$
3. Omomorfismi suriettivi $\varphi: M \rightarrow N$

Dimostrazione: (1 $\rightarrow$ 2) Data una congruenza $\equiv$, definiamo $M/\equiv = \{[a] : a \in M\}$ con operazione $[a][b] = [ab]$. Questa è ben definita per la proprietà di compatibilità.

(2 $\rightarrow$ 3) Il morfismo canonico $\pi: M \rightarrow M/\equiv$ definito da $\pi(a) = [a]$ è un omomorfismo suriettivo.

(3 $\rightarrow$ 1) Dato $\varphi: M \rightarrow N$ suriettivo, definiamo $a \equiv b \Leftrightarrow \varphi(a) = \varphi(b)$. Questa è una congruenza perché:

$$a \equiv b \wedge c \equiv d \Rightarrow \varphi(a) = \varphi(b) \wedge \varphi(c) = \varphi(d)$$

$$\Rightarrow \varphi(ac) = \varphi(a)\varphi(c) = \varphi(b)\varphi(d) = \varphi(bd)$$

$$\Rightarrow ac \equiv bd \quad \blacksquare$$

1.2.2 Congruenze di Indice Finito

Definizione 1.2.2 (Indice di una congruenza):* L'indice di una congruenza $\equiv$ su M è la cardinalità del quoziente $M/\equiv$.

Teorema 1.2.2 (Teorema di Myhill-Nerode esteso):* Sia $L \subseteq \Sigma^*$ un linguaggio. Le seguenti sono equivalenti:

1. L è riconosciuto da un DFA con n stati
2. La congruenza di Myhill-Nerode $\equiv_L$ ha indice n
3. Esiste un monoide M con $|M| = n$ e un omomorfismo $\varphi: \Sigma^* \rightarrow M$ tale che $L = \varphi^{-1}(\varphi(L))$

Dimostrazione: $(1 \rightarrow 2)$ Già dimostrato nel teorema 1.3.1 del documento principale.

$(2 \rightarrow 3)$ Sia $M = \Sigma^*/\equiv_L$ il monoide sintattico di L , e $\varphi: \Sigma^* \rightarrow M$ il morfismo canonico. Allora $|M| = n$ per ipotesi. Inoltre:

$$w \in L \Leftrightarrow [w]_L \in \varphi(L) \Leftrightarrow \varphi(w) \in \varphi(L) \Leftrightarrow w \in \varphi^{-1}(\varphi(L))$$

$(3 \rightarrow 1)$ Costruiamo un DFA $A = (M, \Sigma, \delta, \varphi(\epsilon), F)$ dove:

- $\delta(m, a) = m \cdot \varphi(a)$
- $F = \{m \in M : m \in \varphi(L)\}$

Per induzione su $|w|$, $\delta^*(\varphi(\epsilon), w) = \varphi(w)$. Quindi:

$$w \in L(A) \Leftrightarrow \delta^*(\varphi(\epsilon), w) \in F \Leftrightarrow \varphi(w) \in \varphi(L) \Leftrightarrow w \in L$$

I

1.2.3 Congruenze e Varietà di Linguaggi

Definizione 1.2.3 (Varietà di linguaggi):* Una classe V di linguaggi è una varietà se:

1. È chiusa sotto operazioni booleane ($\cup$, $\cap$, complemento)
2. È chiusa sotto quozienti: se $L \in V$, allora $L/w \in V$ per ogni w
3. È chiusa sotto omomorfismi inversi
4. Contiene solo linguaggi riconoscibili

Teorema 1.2.3 (Teorema di Eilenberg):* Esiste una corrispondenza biunivoca tra varietà di linguaggi e pseudovarietà di monoidi finiti.

Nota: La dimostrazione completa richiede teoria delle categorie avanzata e va oltre lo scopo di questa sezione, ma è fondamentale nella teoria algebrica dei linguaggi formali.

1.2.4 Raffinamento di Congruenze

Definizione 1.2.4 (Raffinamento):* Una congruenza $\equiv_1$ raffina $\equiv_2$ (scritto $\equiv_1 \subseteq \equiv_2$) se:

$$\forall x,y: x \equiv_1 y \Rightarrow x \equiv_2 y$$

Teorema 1.2.4 (Reticolo delle congruenze):* L'insieme delle congruenze su un monoide M , ordinato per raffinamento, forma un reticolo completo.

Dimostrazione: Per ogni famiglia di congruenze $\{\equiv_i\}_{i \in I}$:

Infimo: $\bigcap_i \equiv_i$ è chiaramente una congruenza (intersezione di relazioni di equivalenza compatibili).

Supremo: Definiamo $\equiv$ come la chiusura transitiva di $\bigcup_i \equiv_i$. Verifichiamo che è una congruenza:

- È una relazione di equivalenza (per costruzione)

Compatibilità: se $x \equiv y$ e $z \equiv w$, esistono catene:

$$x \equiv_{\{i_1\}} x_1 \equiv_{\{i_2\}} \dots \equiv_{\{i_k\}} yz \equiv_{\{j_1\}} z_1 \equiv_{\{j_2\}} \dots \equiv_{\{j_l\}} w$$

Quindi:

$$xz \equiv_{\{i_1\}} x_1z \equiv_{\{j_1\}} x_1z_1 \equiv_{\{i_2\}} x_2z_1 \dots \equiv yw \quad \blacksquare$$

1.3 Teoria delle Parole: Proprietà Combinatorie

1.3.1 Lemma di Fine-Wilf

Teorema 1.3.1 (Lemma di Fine-Wilf):* Sia w una stringa che ha periodo p e periodo q . Se $|w| \geq p + q - \gcd(p,q)$, allora w ha periodo $\gcd(p,q)$.

Dove diciamo che w ha periodo p se $w[i] = w[i+p]$ per ogni i tale che entrambi gli indici sono definiti.

Dimostrazione: Sia $d = \gcd(p,q)$. Esistono interi a,b tali che $ap + bq = d$ (identità di Bézout).

Per ogni i con $0 \leq i < |w| - d$, dobbiamo mostrare che $w[i] = w[i+d]$.

Osserviamo che $i + d = i + ap + bq$. Possiamo esprimere questo come una sequenza di passi di dimensione p e q . Poiché $|w| \geq p + q - d$, abbiamo spazio sufficiente per applicare la periodicità in p e q ripetutamente per mostrare $w[i] = w[i+d]$. $\blacksquare$

Corollario 1.3.1:* Se due stringhe u e v commutano ($uv = vu$), allora esiste una stringa t tale che $u,v \in t^*$.

Dimostrazione: Se $uv = vu$, allora la stringa uv ha periodo $|u|$ e periodo $|v|$. Per Fine-Wilf, ha periodo $\gcd(|u|,|v|)$. Sia t la stringa di lunghezza $\gcd(|u|,|v|)$ che è prefisso comune di u e v . Allora u,v sono potenze di t . $\blacksquare$

1.3.2 Teorema di Lyndon-Schützenberger

Definizione 1.3.1 (Stringa primitiva):* Una stringa w è primitiva se non è una potenza propria, ovvero $w \neq v^n$ per $n \geq 2$ e $v \in \Sigma^*$.

Teorema 1.3.2 (Lyndon-Schützenberger):* Se x, y, z sono stringhe primitive e $x^p = y^q = z^r$ con $p, q, r \geq 2$, allora $x = y = z$.

Dimostrazione (Schema): Assumiamo senza perdita di generalità $|x| \leq |y| \leq |z|$.

Caso 1: Se $|x| = |y| = |z|$, allora $x = y = z$ immediatamente.

Caso 2: Se $|x| < |y|$, consideriamo i prefissi di lunghezza $|x|$ in x^p e y^q . Poiché $x^p = y^q$, il prefisso di lunghezza $|x|$ di y^q deve essere x . Ma y è primitivo, quindi questo implica vincoli sulla struttura che portano a contraddizione o $x = y$.

La dimostrazione completa richiede un'analisi casistica accurata basata sulle relazioni tra le lunghezze. ■

1.3.3 Parole di Sturmian e Morfismi

Definizione 1.3.2 (Sequenza di Sturmian):* Una sequenza infinita binaria s è Sturmiana se ha esattamente $n+1$ fattori distinti di lunghezza n per ogni $n \geq 0$.

Teorema 1.3.3 (Caratterizzazione delle sequenze Sturmiane):* Una sequenza binaria infinita è Sturmiana se e solo se è aperiodica e ha complessità fattoriale minima tra le sequenze aperiodiche.

Teorema 1.3.4 (Morfismo di Fibonacci):* Il morfismo $\varphi: \{a, b\}^* \rightarrow \{a, b\}^*$ definito da:

$$\varphi(a) = ab$$

$$\varphi(b) = a$$

genera la sequenza di Fibonacci come punto fisso: $\varphi^\infty(a) = abaababaabaab\dots$

Questa è una sequenza Sturmiana.

1.4 Algebre Libere e Costruzioni Universali

1.4.1 Algebre Universali

Definizione 1.4.1 (Algebra universale):* Un'algebra $A = (A, \{f_i\}_{i \in I})$ consiste di:

- Un carrier set A
- Una famiglia di operazioni $f_i: A^{n_i} \rightarrow A$

Definizione 1.4.2 (Termini):* L'insieme dei termini $T(X)$ su un insieme di variabili X è il più piccolo insieme tale che:

- $X \subseteq T(X)$
- Se $t_1, \dots, t_n \in T(X)$, allora $f(t_1, \dots, t_n) \in T(X)$ per ogni f di arità n

Teorema 1.4.1 (Algebra libera):* Per ogni signature Σ e insieme X , esiste un'algebra libera $F(X)$ tale che:

$$\text{Hom}(F(X), A) \cong \text{Set}(X, U(A))$$

dove U è il funtore oblio che dimentica la struttura algebrica.

1.4.2 Monoidi come Algebre

Osservazione 1.4.1:* Un monoide è un'algebra con:

- Un'operazione binaria $\cdot : M \times M \rightarrow M$
- Una costante $e: M^0 \rightarrow M$

soddisfacenti gli assiomi di associatività e identità.

Teorema 1.4.2 (Varietà di monoidi):* La classe di tutti i monoidi forma una varietà (nel senso delle algebre universali), definita dagli assiomi:

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$

$$e \cdot x = x$$

$$x \cdot e = x$$

1.4.3 Prodotti Liberi di Monoidi

Definizione 1.4.3 (Prodotto libero):* Dati monoidi M_1 e M_2 , il loro prodotto libero $M_1 * M_2$ è il monoide generato da $M_1 \cup M_2$ senza relazioni aggiuntive oltre a quelle in M_1 e M_2 .

Teorema 1.4.3 (Proprietà universale del prodotto libero):* Per ogni coppia di omomorfismi $\varphi_1: M_1 \rightarrow N$ e $\varphi_2: M_2 \rightarrow N$, esiste un unico omomorfismo $\varphi: M_1 * M_2 \rightarrow N$ tale che φ estende φ_1 e φ_2 .

Esempio 1.4.1:* Il prodotto libero $\mathbb{Z} * \mathbb{Z}$ è il gruppo libero su due generatori, ma $\mathbb{N} * \mathbb{N}$ (come monoidi) è diverso: ogni elemento è una sequenza alternata di elementi non-identità da $\mathbb{N}_1$ e $\mathbb{N}_2$.

1.5 Teoremi Avanzati sulle Stringhe

1.5.1 Teorema di Levi

Teorema 1.5.1 (Lemma di Levi):* Date stringhe u, v, x, y tali che $uv = xy$, esiste z tale che una delle seguenti vale:

1. $u = xz$ e $v = zy$
2. $x = uz$ e $v = zy$
3. $u = x$ e $v = y$

Dimostrazione: Consideriamo tre casi basati sulle lunghezze:

Caso 1: $|u| = |x|$. Allora $u = x$ e $v = y$.

Caso 2: $|u| < |x|$. Allora $x = uz$ per qualche z , e $uv = uzy$, quindi $v = zy$.

Caso 3: $|u| > |x|$. Allora $u = xz$ per qualche z , e $xzv = xy$, quindi $zv = y$ e $v = zy$. ■

Corollario 1.5.1:* Se $uv = vu$, allora esiste t tale che $u, v \in t^*$ (già dimostrato come applicazione di Fine-Wilf).

1.5.2 Fattorizzazioni Canoniche

Definizione 1.5.1 (Fattorizzazione di Lyndon):* Ogni stringa w può essere univocamente fattorizzata come $w = l_1 l_2 \dots l_k$ dove ogni l_i è una parola di Lyndon e $l_1 \geq l_2 \geq \dots \geq l_k$ in ordine lessicografico.

Una parola di Lyndon è una stringa che è strettamente minore (lessicograficamente) di ogni suo suffisso proprio.

Teorema 1.5.2 (Esistenza e unicità della fattorizzazione di Lyndon):* Ogni stringa ammette una unica fattorizzazione di Lyndon.

Dimostrazione (Schema): Esistenza: Per induzione sulla lunghezza. Data w , se w è di Lyndon, fatto. Altrimenti, w ha un suffisso proprio lessicograficamente minore. Sia $w = uv$ dove v è il più lungo tale suffisso. Allora u è di Lyndon, e applichiamo induzione a v .

Unicità: Supponiamo due fattorizzazioni $w = l_1 \dots l_k = m_1 \dots m_r$. Se $l_1 = m_1$, procediamo per induzione. Se $l_1 \neq m_1$, assumiamo $|l_1| < |m_1|$. Allora $m_1 = l_1 \dots l_i v$ per qualche i e suffisso proprio v di l_{i+1} . Ma questo contraddirebbe la proprietà di Lyndon di m_1 . ■

1.5.3 Teoremi di Periodicità

Definizione 1.5.2 (Periodo):* Un periodo di una stringa w è un intero p tale che $w[i] = w[i+p]$ per ogni i valido.

Teorema 1.5.3 (Teorema di periodicità forte):* Se w ha periodi p e q con $\gcd(p, q) = 1$ e $|w| \geq p + q - 2$, allora w ha periodo 1 (è una potenza di un singolo carattere).

Teorema 1.5.4 (Critical factorization theorem):* Ogni stringa w ha una factorization $w = uv$ tale che:

1. Il periodo locale di uv nella posizione $|u|$ è uguale al periodo globale minimo di w

2. Tale posizione è unica e può essere calcolata in tempo lineare

Questo teorema è fondamentale per algoritmi di pattern matching ottimali.

Capitolo 2: Teoria degli Automi

2.1 Automi a Stati Finiti Deterministici

Definizione 2.1.1 (Automa a stati finiti deterministico): Un automa a stati finiti deterministico (DFA) è una quintupla $A = (Q, \Sigma, \delta, q_0, F)$ dove:

- Q è un insieme finito di stati
- Σ è un alfabeto finito (l'alfabeto di input)
- $\delta : Q \times \Sigma \rightarrow Q$ è la funzione di transizione
- $q_0 \in Q$ è lo stato iniziale
- $F \subseteq Q$ è l'insieme degli stati finali (o accettanti)

Definizione 2.1.2 (Funzione di transizione estesa): La funzione di transizione estesa $\delta^* : Q \times \Sigma^* \rightarrow Q$ è definita ricorsivamente da:

- $\delta^*(q, \varepsilon) = q$ per ogni $q \in Q$
- $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$ per ogni $q \in Q, w \in \Sigma^*, a \in \Sigma$

Definizione 2.1.3 (Linguaggio accettato): Il linguaggio accettato (o riconosciuto) da un DFA $A = (Q, \Sigma, \delta, q_0, F)$ è: $L(A) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F\}$

Definizione 2.1.4 (Linguaggio regolare): Un linguaggio $L \subseteq \Sigma^*$ è detto regolare se esiste un DFA A tale che $L = L(A)$.

Teorema 2.1.1 (Chiusura dei linguaggi regolari sotto unione): Se L_1 e L_2 sono linguaggi regolari, allora $L_1 \cup L_2$ è regolare.

Dimostrazione: Siano $A_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$ e $A_2 = (Q_2, \Sigma, \delta_2, q_0^2, F_2)$ DFA che riconoscono rispettivamente L_1 e L_2 . Costruiamo un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce $L_1 \cup L_2$ mediante la costruzione del prodotto:

- $Q = Q_1 \times Q_2$
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$ per ogni $(q_1, q_2) \in Q$ e $a \in \Sigma$
- $q_0 = (q_0^1, q_0^2)$
- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

Per induzione su $|w|$, si dimostra che $\delta^*((q_0^1, q_0^2), w) = (\delta_1^*(q_0^1, w), \delta_2^*(q_0^2, w))$.

Quindi $w \in L(A)$ se e solo se $\delta^*((q_0^1, q_0^2), w) \in F$, cioè se e solo se $\delta_1^*(q_0^1, w) \in F_1$ oppure $\delta_2^*(q_0^2, w) \in F_2$, il che equivale a $w \in L_1 \cup L_2$. ■

Teorema 2.1.2 (Chiusura dei linguaggi regolari sotto intersezione): Se L_1 e L_2 sono linguaggi regolari, allora $L_1 \cap L_2$ è regolare.

Dimostrazione: La costruzione è simile a quella del teorema precedente, ma con $F = F_1 \times F_2$. Così $w \in L(A)$ se e solo se $\delta_1^*(q_0^1, w) \in F_1$ e $\delta_2^*(q_0^2, w) \in F_2$, cioè $w \in L_1 \cap L_2$. ■

Teorema 2.1.3 (Chiusura dei linguaggi regolari sotto complemento): Se L è un linguaggio regolare su Σ , allora $\Sigma^* \setminus L$ è regolare.

Dimostrazione: Sia $A = (Q, \Sigma, \delta, q_0, F)$ un DFA che riconosce L . Costruiamo $A' = (Q, \Sigma, \delta, q_0, Q \setminus F)$. Allora $w \in L(A')$ se e solo se $\delta^*(q_0, w) \in Q \setminus F$, cioè se e solo se $\delta^*(q_0, w) \notin F$, il che equivale a $w \notin L(A) = L$. Quindi $L(A') = \Sigma^* \setminus L$. ■

Teorema 2.1.4 (Chiusura dei linguaggi regolari sotto concatenazione): Se L_1 e L_2 sono linguaggi regolari, allora $L_1 L_2$ è regolare.

Dimostrazione: Questa dimostrazione richiede l'uso di automi a stati finiti non deterministici (NFA), che introdurremo nella prossima sezione. La dimostrazione viene rimandata al Teorema 2.2.3. ■

2.2 Automi a Stati Finiti Non Deterministici

Definizione 2.2.1 (Automa a stati finiti non deterministico): Un automa a stati finiti non deterministico (NFA) è una quintupla $N = (Q, \Sigma, \Delta, Q_0, F)$ dove:

- Q è un insieme finito di stati
- Σ è un alfabeto finito
- $\Delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ è la funzione di transizione
- $Q_0 \subseteq Q$ è l'insieme degli stati iniziali
- $F \subseteq Q$ è l'insieme degli stati finali

Definizione 2.2.2 (Funzione di transizione estesa per NFA): La funzione di transizione estesa $\Delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$ è definita da:

- $\Delta^*(q, \epsilon) = \{q\}$
- $\Delta^*(q, wa) = \bigcup \{\Delta(p, a) : p \in \Delta^*(q, w)\}$

Estendiamo Δ^* a insiemi di stati: per $S \subseteq Q$, $\Delta^*(S, w) = \bigcup \{\Delta^*(q, w) : q \in S\}$.

Definizione 2.2.3 (Linguaggio accettato da un NFA): Il linguaggio accettato da un NFA $N = (Q, \Sigma, \Delta, Q_0, F)$ è: $L(N) = \{w \in \Sigma^* : \Delta^*(Q_0, w) \cap F \neq \emptyset\}$

Teorema 2.2.1 (Equivalenza DFA-NFA): Un linguaggio è accettato da un NFA se e solo se è accettato da un DFA.

Dimostrazione: ($\Rightarrow$) È immediato che ogni DFA è un NFA particolare dove $|Q_0| = 1$ e $|\Delta(q, a)| = 1$ per ogni q e a .

($\Leftarrow$) Sia $N = (Q_N, \Sigma, \Delta_N, Q_0, F_N)$ un NFA. Costruiamo un DFA $D = (Q_D, \Sigma, \delta_D, q_0_D, F_D)$ mediante la costruzione dei sottoinsiemi (subset construction):

- $Q_D = \mathcal{P}(Q_N)$ (l'insieme delle parti di Q_N)
- $\delta_D(S, a) = \Delta_N(S, a) = \bigcup \{\Delta_N(q, a) : q \in S\}$ per ogni $S \subseteq Q_N$ e $a \in \Sigma$

- $q_{0_D} = Q_0$
- $F_D = \{S \subseteq Q_N : S \cap F_N \neq \emptyset\}$

Dimostriamo per induzione su $|w|$ che $\delta_D^*(q_{0_D}, w) = \Delta_N^*(Q_0, w)$.

Base: Se $w = \varepsilon$, allora $\delta_D^*(q_{0_D}, \varepsilon) = q_{0_D} = Q_0 = \Delta_N^*(Q_0, \varepsilon)$.

Passo: Sia $w = ua$ con $|u| = n$. Per ipotesi induttiva, $\delta_D^*(q_{0_D}, u) = \Delta_N^*(Q_0, u)$. Allora:
 $\delta_D^*(q_{0_D}, ua) = \delta_D(\delta_D^*(q_{0_D}, u), a) = \delta_D(\Delta_N^*(Q_0, u), a) = \Delta_N(\Delta_N^*(Q_0, u), a) = \Delta_N^*(Q_0, ua)$

Quindi $w \in L(D)$ se e solo se $\delta_D^*(q_{0_D}, w) \in F_D$, cioè se e solo se $\Delta_N^*(Q_0, w) \cap F_N \neq \emptyset$, che equivale a $w \in L(N)$. ■

Definizione 2.2.4 (NFA con ε -transizioni): Un NFA con ε -transizioni (ε -NFA) è una quintupla $N_ \varepsilon = (Q, \Sigma, \Delta, Q_0, F)$ dove $\Delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$.

Definizione 2.2.5 (ε -chiusura): Per un insieme di stati $S \subseteq Q$, la ε -chiusura di S , denotata $\varepsilon\text{-CLOSURE}(S)$, è il più piccolo insieme che contiene S ed è chiuso sotto le ε -transizioni:
 $\varepsilon\text{-CLOSURE}(S) = \{q \in Q : \text{esiste un cammino di } \varepsilon\text{-transizioni da qualche stato in } S \text{ a } q\}$

Formalmente, $\varepsilon\text{-CLOSURE}(S)$ può essere definito come il limite della sequenza:

- $S_0 = S$
- $S_{i+1} = S_i \cup \{q \in Q : \text{esiste } p \in S_i \text{ con } q \in \Delta(p, \varepsilon)\}$

Teorema 2.2.2 (Eliminazione delle ε -transizioni): Per ogni ε -NFA esiste un NFA equivalente senza ε -transizioni.

Dimostrazione: Sia $N_ \varepsilon = (Q, \Sigma, \Delta_ \varepsilon, Q_0, F)$ un ε -NFA. Costruiamo un NFA $N = (Q, \Sigma, \Delta, Q'_0, F')$ dove:

- $\Delta(q, a) = \varepsilon\text{-CLOSURE}(\Delta_ \varepsilon(\varepsilon\text{-CLOSURE}(\{q\}), a))$ per ogni $q \in Q$ e $a \in \Sigma$
- $Q'_0 = \varepsilon\text{-CLOSURE}(Q_0)$
- $F' = \{q \in Q : \varepsilon\text{-CLOSURE}(\{q\}) \cap F \neq \emptyset\}$

Si può dimostrare per induzione che $\Delta^*(Q'_0, w) = \varepsilon\text{-CLOSURE}(\Delta_ \varepsilon^*(Q_0, w))$ per ogni $w \in \Sigma^*$. Quindi $L(N) = L(N_ \varepsilon)$. ■

Teorema 2.2.3 (Chiusura dei linguaggi regolari sotto concatenazione - dimostrazione completa): Se L_1 e L_2 sono linguaggi regolari, allora L_1L_2 è regolare.

Dimostrazione: Siano $N_1 = (Q_1, \Sigma, \Delta_1, \{q_0^1\}, F_1)$ e $N_2 = (Q_2, \Sigma, \Delta_2, \{q_0^2\}, F_2)$ NFA che riconoscono L_1 e L_2 rispettivamente (assumiamo per semplicità $Q_1 \cap Q_2 = \emptyset$). Costruiamo un ε -NFA $N = (Q, \Sigma, \Delta, \{q_0^1\}, F_2)$ dove:

- $Q = Q_1 \cup Q_2$
- Per ogni $q \in Q$ e $a \in \Sigma$:
 - Se $q \in Q_1 \setminus F_1$, allora $\Delta(q, a) = \Delta_1(q, a)$
 - Se $q \in F_1$, allora $\Delta(q, a) = \Delta_1(q, a)$ e $\Delta(q, \varepsilon) = \{q_0^2\}$

- Se $q \in Q_2$, allora $\Delta(q, a) = \Delta_2(q, a)$

Intuitivamente, N simula N_1 finché non raggiunge uno stato finale di F_1 , momento in cui può non-deterministicamente passare allo stato iniziale di N_2 tramite una ε -transizione.

Una stringa w è accettata da N se e solo se può essere decomposta come $w = w_1 w_2$ dove N_1 accetta w_1 (raggiungendo uno stato in F_1) e da lì N_2 accetta w_2 . Quindi $L(N) = L_1 L_2$. Per i teoremi precedenti, $L_1 L_2$ è regolare. ■

Teorema 2.2.4 (Chiusura sotto la chiusura di Kleene): Se L è un linguaggio regolare, allora L^* è regolare.

Dimostrazione: Sia $N = (Q, \Sigma, \Delta_N, \{q_0\}, F)$ un NFA che riconosce L . Introduciamo un nuovo stato $q_{\text{new}} \notin Q$ e costruiamo un ε -NFA $N^* = (Q \cup \{q_{\text{new}}\}, \Sigma, \Delta^*, \{q_{\text{new}}\}, \{q_{\text{new}}\})$ dove:

- $\Delta^*(q_{\text{new}}, \varepsilon) = \{q_0\}$
- Per ogni $q \in F$, $\Delta^*(q, \varepsilon)$ contiene q_{new} (oltre alle eventuali ε -transizioni già presenti)
- $\Delta^*(q, a) = \Delta_N(q, a)$ per ogni $q \in Q$ e $a \in \Sigma$

Lo stato q_{new} è sia iniziale che finale, quindi $\varepsilon \in L(N^*)$. Inoltre, ogni volta che N raggiunge uno stato finale, può tornare a q_{new} tramite ε -transizione e ricominciare il riconoscimento. Quindi N^* accetta esattamente le stringhe in L^* . ■

2.3 Minimizzazione di Automi e Teorema di Myhill-Nerode

Definizione 2.3.1 (Stati equivalenti): Due stati $q_1, q_2 \in Q$ di un DFA $A = (Q, \Sigma, \delta, q_0, F)$ sono equivalenti ($q_1 \approx q_2$) se per ogni $w \in \Sigma^*$: $\delta^*(q_1, w) \in F \Leftrightarrow \delta^*(q_2, w) \in F$

Definizione 2.3.2 (Stati distinguibili): Due stati q_1, q_2 sono k -distinguibili se esiste una stringa w con $|w| \leq k$ tale che esattamente uno tra $\delta^*(q_1, w)$ e $\delta^*(q_2, w)$ appartiene a F .

Lemma 2.3.1: La relazione $\approx$ è una congruenza sull'insieme degli stati Q .

Dimostrazione: Verifichiamo che $\approx$ è una relazione di equivalenza: (i) Riflessività: Ovviamente $q \approx q$. (ii) Simmetria: Se $q_1 \approx q_2$, allora per ogni w , $\delta^*(q_1, w) \in F \Leftrightarrow \delta^*(q_2, w) \in F$, quindi anche $q_2 \approx q_1$. (iii) Transitività: Se $q_1 \approx q_2$ e $q_2 \approx q_3$, allora per ogni w : $\delta^*(q_1, w) \in F \Leftrightarrow \delta^*(q_2, w) \in F \Leftrightarrow \delta^*(q_3, w) \in F$ Quindi $q_1 \approx q_3$.

Inoltre, $\approx$ rispetta le transizioni: se $q_1 \approx q_2$, allora per ogni $a \in \Sigma$, $\delta(q_1, a) \approx \delta(q_2, a)$. Infatti, per ogni $w \in \Sigma^*$: $\delta^*(\delta(q_1, a), w) = \delta^*(q_1, aw) \in F \Leftrightarrow \delta^*(q_2, aw) \in F \Leftrightarrow \delta^*(\delta(q_2, a), w) \in F$ ■

Algoritmo 2.3.1 (Algoritmo di minimizzazione di Hopcroft): L'algoritmo costruisce iterativamente la partizione degli stati in classi di equivalenza:

1. Inizializza la partizione $P_0 = \{F, Q \setminus F\}$
2. Per $i = 0, 1, 2, \dots$, finché $P_i \neq P_{i+1}$:
 - Per ogni classe C in P_i e per ogni simbolo $a \in \Sigma$:

- Partiziona C in sottoclassi $C_1, C_2, \dots, C_k$ dove $q_1, q_2 \in C_j$ se e solo se $\delta(q_1, a)$ e $\delta(q_2, a)$ appartengono alla stessa classe di P_i
 - P_{i+1} è la partizione ottenuta raffinando tutte le classi di P_i
- 3. Quando l'algoritmo termina, $P = P_n$ è la partizione degli stati in classi di equivalenza

Teorema 2.3.2 (Correttezza dell'algoritmo di minimizzazione): L'algoritmo di Hopcroft termina e produce la partizione di equivalenza degli stati.

Dimostrazione: Poiché $|Q|$ è finito, la sequenza di partizioni $P_0, P_1, P_2, \dots$ è strettamente raffinante fino a quando non si stabilizza. Ogni P_i è un raffinamento di P_{i-1} , e il numero di classi è limitato da $|Q|$, quindi l'algoritmo termina.

Dimostriamo che quando termina, gli stati nella stessa classe finale sono equivalenti. Sia P la partizione finale. Supponiamo per contraddizione che esistano q_1, q_2 nella stessa classe $C \in P$ ma $q_1 \neq q_2$. Allora esiste $w = a_1 a_2 \dots a_n$ di lunghezza minima tale che $\delta^*(q_1, w) \in F$ e $\delta^*(q_2, w) \notin F$ (o viceversa).

Se $n = 0$, allora $w = \varepsilon$ e $q_1 \in F, q_2 \notin F$, contraddicendo il fatto che entrambi sono in C (poiché P_0 separa F da $Q \setminus F$).

Se $n > 0$, sia $p_1 = \delta(q_1, a_1)$ e $p_2 = \delta(q_2, a_1)$. Per la minimalità di w , p_1 e p_2 devono essere in classi diverse alla fine dell'algoritmo. Ma allora q_1 e q_2 sarebbero stati separati durante il raffinamento rispetto al simbolo a_1 , contraddicendo il fatto che sono nella stessa classe finale. ■

Teorema 2.3.3 (DFA minimo): Sia L un linguaggio regolare. Il DFA minimo che riconosce L è unico a meno di isomorfismo e ha tanti stati quante sono le classi di equivalenza di $\equiv_L$.

Dimostrazione: Sia $A = (Q, \Sigma, \delta, q_0, F)$ un DFA minimo per L . Definiamo $\varphi : Q \rightarrow \{[w]_L : w \in \Sigma^*\}$ ponendo $\varphi(q) = [w]_L$ dove w è una qualunque stringa tale che $\delta^*(q_0, w) = q$.

φ è ben definita: se $\delta^*(q_0, w_1) = \delta^*(q_0, w_2) = q$, allora $w_1 \equiv_L w_2$. Infatti, per ogni $z \in \Sigma^*$: $w_1 z \in L \Leftrightarrow \delta^*(q_0, w_1 z) \in F \Leftrightarrow \delta^*(\delta^*(q_0, w_1), z) \in F \Leftrightarrow \delta^*(q, z) \in F \Leftrightarrow \delta^*(\delta^*(q_0, w_2), z) \in F \Leftrightarrow \delta^*(q_0, w_2 z) \in F \Leftrightarrow w_2 z \in L$

φ è iniettiva: se $\varphi(q_1) = \varphi(q_2)$, allora esistono w_1, w_2 con $\delta^*(q_0, w_1) = q_1, \delta^*(q_0, w_2) = q_2$, e $w_1 \equiv_L w_2$. Ma allora q_1 e q_2 sono equivalenti in A , e poiché A è minimo, $q_1 = q_2$.

φ è suriettiva: per ogni classe $[w]_L$, lo stato $\delta^*(q_0, w)$ mappa a $[w]_L$ sotto φ .

Quindi $|Q| = |\{[w]_L : w \in \Sigma^*\}|$, cioè il numero di stati del DFA minimo è esattamente l'indice di $\equiv_L$. L'unicità segue dal fatto che qualsiasi DFA minimo deve avere questa struttura. ■

2.4 Automi Bidirezionali e Automi a Due Vie

Definizione 2.4.1 (Automa a due vie deterministico - 2DFA): Un automa a due vie deterministico è una sestupla $M = (Q, \Sigma, \vdash, \dashv, \delta, q_0, F)$ dove:

- Q è l'insieme finito degli stati

- Σ è l'alfabeto di input
- $\vdash$ è il marcatore di inizio (left endmarker)
- $\dashv$ è il marcatore di fine (right endmarker)
- $\delta: Q \times (\Sigma \cup \{\vdash, \dashv\}) \rightarrow Q \times \{L, R, S\}$ è la funzione di transizione che restituisce il nuovo stato e la direzione di movimento (Left, Right, Stay)
- $q_0 \in Q$ è lo stato iniziale
- $F \subseteq Q$ è l'insieme degli stati finali

Teorema 2.4.1 (Equivalenza 2DFA-DFA): Per ogni 2DFA esiste un DFA equivalente che riconosce lo stesso linguaggio.

Dimostrazione (Schema costruttivo): La dimostrazione procede attraverso una simulazione mediante "through configurations". Per ogni coppia di stati (p, q) , costruiamo una funzione $f_{\{p, q\}}: \Sigma^* \rightarrow Q$ che rappresenta lo stato raggiunto leggendo l'input da p a q .

Definiamo l'insieme delle configurazioni crossings: $C = \{(p, q) \in Q \times Q : \exists w \in \Sigma^*, \text{ il 2DFA parte da } p \text{ e raggiunge } q \text{ dopo aver attraversato ogni posizione dell'input un numero finito di volte}\}$

Il DFA simulatore ha stati corrispondenti alle funzioni di crossing. Poiché il numero di tali funzioni è finito (al massimo $|Q|^{\wedge}|Q|^2$), il DFA ha un numero finito di stati.

Complessità: Se il 2DFA ha n stati, il DFA equivalente può avere fino a $n^{\wedge}\{O(n^2)\}$ stati nel caso peggiore. ■

Teorema 2.4.2 (Caratterizzazione dei crossing sequences): Un linguaggio L è regolare se e solo se per ogni stringa $w \in \Sigma^*$ esiste un bound k tale che ogni computazione accettante attraversa ogni posizione al massimo k volte.

2.5 Teorema di Myhill-Nerode: Forma Estesa

Definizione 2.5.1 (Kernel di un omomorfismo): Dato un omomorfismo $\varphi: \Sigma^* \rightarrow M$ in un monoide finito M , il kernel di φ è la congruenza: $x \equiv_{\varphi} y \Leftrightarrow \varphi(x) = \varphi(y)$

Teorema 2.5.1 (Caratterizzazione algebrica dei linguaggi regolari): Le seguenti condizioni sono equivalenti per un linguaggio $L \subseteq \Sigma^*$:

1. L è riconosciuto da un DFA
2. L è l'unione di classi di equivalenza di una congruenza di indice finito
3. L è il kernel di un omomorfismo in un monoide finito
4. Il monoide sintattico di L è finito

Dimostrazione: $(1 \Rightarrow 2)$: La congruenza di Myhill-Nerode $\equiv_L$ ha indice finito uguale al numero di stati del DFA minimo.

$(2 \Rightarrow 3)$: Sia $\equiv$ una congruenza di indice finito tale che L è unione di classi di $\equiv$. Definiamo $M = \Sigma^* / \equiv$ (il monoide quoziente) e $\varphi: \Sigma^* \rightarrow M$ l'omomorfismo canonico. Allora $\ker(\varphi) = \equiv$ e $L = \varphi^{-1}(\varphi(L))$.

$(3 \Rightarrow 4)$: Il monoide sintattico $M_L = \Sigma^* / \equiv_L$ è un quoziente di M , quindi finito.

(4 $\Rightarrow$ 1): Se M_L è finito, costruiamo un DFA con stati = M_L , transizioni date dalla moltiplicazione nel monoide. **I**

Definizione 2.5.2 (Monoide sintattico): Il monoide sintattico di L è: $M_L = \{[w]_{\equiv_L} : w \in \Sigma^*\}$ con operazione $[u] \cdot [v] = [uv]$.

Teorema 2.5.2 (Proprietà del monoide sintattico):

1. M_L divide ogni monoide che riconosce L
2. $|M_L|$ = numero minimo di stati di un DFA per L
3. M_L è calcolabile algebricamente dato un DFA per L

2.6 Automi con Output: Macchine di Mealy e Moore

Definizione 2.6.1 (Macchina di Mealy): Una macchina di Mealy è una sestupla $M = (Q, \Sigma, \Lambda, \delta, \lambda, q_0)$ dove:

- Q, Σ, δ, q_0 come per DFA
- Λ è l'alfabeto di output
- $\lambda: Q \times \Sigma \rightarrow \Lambda$ è la funzione di output

L'output per una stringa $w = a_1 \dots a_n$ è: $\lambda^*(q_0, w) = \lambda(q_0, a_1)\lambda(\delta(q_0, a_1), a_2) \dots \lambda(\delta^*(q_0, a_1 \dots a_{n-1}), a_n)$

Definizione 2.6.2 (Macchina di Moore): Una macchina di Moore è una sestupla $M = (Q, \Sigma, \Lambda, \delta, \lambda, q_0)$ dove $\lambda: Q \rightarrow \Lambda$.

L'output per w è: $\lambda^*(q_0, w) = \lambda(q_0)\lambda(\delta(q_0, a_1))\lambda(\delta(q_0, a_1 a_2)) \dots \lambda(\delta^*(q_0, w))$

Teorema 2.6.1 (Equivalenza Mealy-Moore): Per ogni macchina di Mealy esiste una macchina di Moore equivalente (a meno di un simbolo di ritardo) e viceversa.

Dimostrazione: Mealy $\rightarrow$ Moore: Dato $M_{\text{Mealy}} = (Q, \Sigma, \Lambda, \delta, \lambda, q_0)$, costruiamo M_{Moore} con:

- $Q' = Q \times \Lambda$
- $\lambda'((q, a)) = a$
- $\delta'((q, a), \sigma) = (\delta(q, \sigma), \lambda(q, \sigma))$

Moore $\rightarrow$ Mealy: Immediato ponendo $\lambda_{\text{Mealy}}(q, \sigma) = \lambda_{\text{Moore}}(\delta(q, \sigma))$. **I**

Capitolo 2 Espanso: Teoria Avanzata degli Automi

2.7 Automi Probabilistici: Teoria Completa

2.7.1 Hidden Markov Models: Formalizzazione Rigorosa

*Definizione 2.7.1 (HMM - definizione estesa):**

$$\text{HMM} = (Q, \Sigma, A, B, \pi, F)$$

dove:

- $Q = \{q_1, \dots, q_n\}$: stati nascosti
- $\Sigma = \{\sigma_1, \dots, \sigma_m\}$: alfabeto osservabile

$A \in \mathbb{R}^{n \times n}$: matrice di transizione

$$A_{ij} = P(q_{t+1} = j \mid q_t = i) \text{ vincoli: } A_{ij} \geq 0, \sum_j A_{ij} = 1$$

•

$B \in \mathbb{R}^{n \times m}$: matrice di emissione

$$B_{ik} = P(O_t = \sigma_k \mid q_t = i) \text{ vincoli: } B_{ik} \geq 0, \sum_k B_{ik} = 1$$

•

$\pi \in \mathbb{R}^n$: distribuzione iniziale

$$\pi_i = P(q_0 = i) \text{ vincoli: } \pi_i \geq 0, \sum_i \pi_i = 1$$

•

- $F \subseteq Q$: stati finali (opzionale)

*Teorema 2.7.1 (Problema di Evaluation - formale):**

Input: HMM $\lambda = (A, B, \pi)$, Osservazioni $O = o_1 o_2 \dots o_T$

Output: $P(O \mid \lambda)$

Formula diretta (esponenziale):

$$P(O \mid \lambda) = \sum_{Q=q_1 \dots q_T} P(O, Q \mid \lambda)$$

$$= \sum_{Q} \pi_{q_1} B_{q_1, o_1} \prod_{t=1}^{T-1} A_{q_t, q_{t+1}} B_{q_{t+1}, o_{t+1}}$$

Complessità: $O(T \cdot N^T)$ - impraticabile

*Algoritmo 2.7.1 (Forward Algorithm - ottimizzato):**

Definizione della forward variable:

$$\alpha_t(i) = P(o_1 o_2 \dots o_t, q_t = i \mid \lambda)$$

Ricorsione:

Inizializzazione:

$$\alpha_1(i) = \pi_i \cdot B_{\{i, o_1\}}, \quad 1 \leq i \leq N$$

Induzione:

$$\alpha_{t+1}(j) = [\sum_{i=1}^N \alpha_t(i) \cdot A_{\{ij\}}] \cdot B_{\{j, o_{t+1}\}}$$

Terminazione:

$$P(O \mid \lambda) = \sum_{i=1}^N \alpha_T(i)$$

Complessità: $O(T \cdot N^2)$

Dimostrazione di correttezza:

Per induzione su t :

Base ($t=1$):

$$\alpha_1(i) = P(o_1, q_1=i \mid \lambda) = \pi_i \cdot B_{\{i, o_1\}} \quad \checkmark$$

Passo induttivo: Assumiamo $\alpha_t(i) = P(o_1 \dots o_t, q_t=i \mid \lambda)$.

Vogliamo mostrare:

$$\alpha_{t+1}(j) = P(o_1 \dots o_t o_{t+1}, q_{t+1}=j \mid \lambda)$$

Per la legge della probabilità totale:

$$P(o_1 \dots o_{t+1}, q_{t+1}=j \mid \lambda)$$

$$= \sum_{i=1}^N P(o_1 \dots o_{t+1}, q_t=i, q_{t+1}=j \mid \lambda)$$

$$= \sum_{i=1}^N P(o_1 \dots o_t, q_t=i \mid \lambda) \cdot P(q_{t+1}=j \mid q_t=i) \cdot P(o_{t+1} \mid q_{t+1}=j)$$

$$= \sum_{i=1}^N \alpha_t(i) \cdot A_{\{ij\}} \cdot B_{\{j, o_{t+1}\}}$$

$$= [\sum_{i=1}^N \alpha_t(i) \cdot A_{ij}] \cdot B_{j,o_{t+1}} \quad \blacksquare$$

*Algoritmo 2.7.2 (Backward Algorithm):**

Definizione della backward variable:

$$\beta_t(i) = P(o_{t+1} \dots o_T \mid q_t = i, \lambda)$$

Ricorsione (all'indietro):

Inizializzazione:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N$$

Induzione (t da T-1 a 1):

$$\beta_t(i) = \sum_{j=1}^N A_{ij} \cdot B_{j,o_{t+1}} \cdot \beta_{t+1}(j)$$

Calcolo finale:

$$P(O \mid \lambda) = \sum_{i=1}^N \pi_i \cdot B_{i,o_1} \cdot \beta_1(i)$$

*Teorema 2.7.2 (Forward-Backward consistency):**

$$\forall t, \forall i: \alpha_t(i) \cdot \beta_t(i) = P(O, q_t=i \mid \lambda)$$

Quindi:

$$P(O \mid \lambda) = \sum_{i=1}^N \alpha_t(i) \cdot \beta_t(i) \quad \text{per ogni } t$$

2.7.2 Viterbi Algorithm: Path Ottimo

Problema 2.7.1 (Decoding):

Input: HMM λ , Osservazioni O

Output: $Q^* = \operatorname{argmax}_Q P(Q \mid O, \lambda)$

*Algoritmo 2.7.3 (Viterbi):**

Definizione:

$$\delta_t(i) = \max_{q_1 \dots q_{t-1}} P(q_1 \dots q_{t-1}, q_t=i, o_1 \dots o_t \mid \lambda)$$

Ricorsione:

Inizializzazione:

$$\delta_1(i) = \pi_i \cdot B_{\{i, o_1\}}$$

$$\psi_1(i) = 0$$

Ricorsione ($t = 2 \dots T$):

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) \cdot A_{\{ij\}}] \cdot B_{\{j, o_t\}}$$

$$\psi_t(j) = \operatorname{argmax}_{1 \leq i \leq N} [\delta_{t-1}(i) \cdot A_{\{ij\}}]$$

Terminazione:

$$P^* = \max_{1 \leq i \leq N} \delta_T(i)$$

$$q_{T^*} = \operatorname{argmax}_{1 \leq i \leq N} \delta_T(i)$$

Backtracking ($t = T-1 \dots 1$):

$$q_{t^*} = \psi_{\{t+1\}}(q_{\{t+1\}^*})$$

Complessità: $O(T \cdot N^2)$ *Teorema 2.7.3 (Ottimalità di Viterbi):**Il path Q^* restituito da Viterbi massimizza $P(Q \mid O, \lambda)$

Dimostrazione: Per programmazione dinamica. Supponiamo per assurdo che esista un path Q' con probabilità maggiore. Allora deve esistere un tempo t e stato i tale che il

sub-path ottimale fino a (t,i) in Q' è diverso da quello in Q^* . Ma questo contraddirebbe l'ottimalità locale del sub-path in Q^* calcolato da Viterbi. ■

2.7.3 Baum-Welch Algorithm: Training

Problema 2.7.2 (Learning):

Input: Osservazioni O , struttura HMM

Output: Parametri $\lambda^* = (A^*, B^*, \pi^*)$ che massimizzano $P(O | \lambda)$

*Algoritmo 2.7.4 (Baum-Welch - EM algorithm):**

Definizioni:

$$\begin{aligned}\xi_t(i,j) &= P(q_t=i, q_{t+1}=j | O, \lambda) \\ &= [\alpha_t(i) \cdot A_{ij} \cdot B_{j,o_{t+1}} \cdot \beta_{t+1}(j)] / P(O|\lambda)\end{aligned}$$

$$\begin{aligned}\gamma_t(i) &= P(q_t=i | O, \lambda) \\ &= [\alpha_t(i) \cdot \beta_t(i)] / P(O|\lambda) \\ &= \sum_{j=1}^N \xi_t(i,j)\end{aligned}$$

Iterazione EM:

E-step (calcola statistiche attese):

Calcola $\alpha_t(i)$, $\beta_t(i)$, $\xi_t(i,j)$, $\gamma_t(i)$ usando λ corrente

M-step (aggiorna parametri):

$$\pi'_i = \gamma_1(i)$$

$$A'_{ij} = [\sum_{t=1}^{T-1} \xi_t(i,j)] / [\sum_{t=1}^{T-1} \gamma_t(i)]$$

$$B'_{jk} = [\sum_{t=1, o_t=\sigma_k} \gamma_t(j)] / [\sum_{t=1}^T \gamma_t(j)]$$

*Teorema 2.7.4 (Convergenza di Baum-Welch):**

Baum-Welch converge a un massimo locale di $P(O | \lambda)$

Dimostrazione (Schema): Ogni iterazione EM aumenta (non-strictly) la log-likelihood:

$$\log P(O | \lambda^{(n+1)}) \geq \log P(O | \lambda^{(n)})$$

Questo segue dalla proprietà generale degli algoritmi EM:

$$Q(\lambda' | \lambda) = \sum_Q P(Q | O, \lambda) \log P(O, Q | \lambda')$$

L'M-step massimizza Q , garantendo miglioramento. Poiché il parametro space è compatto e P è continua, la sequenza converge. ■

2.7.4 Applicazioni: Speech Recognition

Modello 2.7.1 (ASR con HMM):

Speech Signal $\rightarrow$ Features (MFCC) $\rightarrow$ HMM $\rightarrow$ Words

Matematicamente:

$$\begin{aligned} W^* &= \operatorname{argmax}_W P(W | O) \\ &= \operatorname{argmax}_W P(O | W) \cdot P(W) / P(O) \\ &= \operatorname{argmax}_W P(O | W) \cdot P(W) \end{aligned}$$

dove:

- $P(O | W)$: Acoustic Model (HMM per ogni fonema/parola)
- $P(W)$: Language Model (n-gram o RNN)

*Teorema 2.7.5 (Decodifica con Beam Search):**

Per HMM grande-scala, Viterbi esatto è impraticabile.

Beam Search mantiene solo top-K ipotesi ad ogni time step.

Complessità: $O(T \cdot K \cdot B)$

dove B = beam width

Garanzia: Se K sufficientemente grande,

approximation ratio $\rightarrow 1$ con alta probabilità

2.8 Automi su Alberi (Tree Automata)

2.8.1 Definizione Formale

*Definizione 2.8.1 (Ranked Alphabet):**

Un ranked alphabet è $\Sigma = \bigcup_{n \geq 0} \Sigma_n$

dove Σ_n = simboli con arità n

*Definizione 2.8.2 (Tree Language):**

T_Σ = insieme di tutti gli alberi finiti su Σ

$T_\Sigma(X)$ = alberi con variabili da X

Definizione induttiva:

- $x \in X \Rightarrow x \in T_\Sigma(X)$

- $f \in \Sigma_n, t_1, \dots, t_n \in T_\Sigma(X) \Rightarrow f(t_1, \dots, t_n) \in T_\Sigma(X)$

*Definizione 2.8.3 (Bottom-Up Tree Automaton):**

$A = (Q, \Sigma, Q_f, \Delta)$

dove:

- Q : stati finiti

- Σ : ranked alphabet
- $Q_f \subseteq Q$: stati finali

Δ : transizioni:

$\Delta \subseteq \bigcup_{n \geq 0} (\Sigma_n \times Q^n \times Q)$ Notazione: $f(q_1, \dots, q_n) \rightarrow q$

•

*Definizione 2.8.4 (Run di Tree Automaton):**

Un run di A su albero t è una funzione:

$r: \text{positions}(t) \rightarrow Q$

tale che:

$\forall$ position p in t con label f e figli a posizioni $p_1, \dots, p_n$:

$f(r(p_1), \dots, r(p_n)) \rightarrow r(p) \in \Delta$

*Definizione 2.8.5 (Linguaggio riconosciuto):**

$L(A) = \{t \in T_\Sigma : \exists \text{ run } r \text{ su } t \text{ con } r(\text{root}) \in Q_f\}$

2.8.2 Proprietà di Chiusura

*Teorema 2.8.1 (Chiusura sotto operazioni booleane):**

I linguaggi regolari di alberi sono chiusi sotto:

- Unione
- Intersezione
- Complemento

Dimostrazione (Unione): Dati $A_1 = (Q_1, \Sigma, F_1, \Delta_1)$ e $A_2 = (Q_2, \Sigma, F_2, \Delta_2)$:

Costruiamo $A = (Q_1 \cup Q_2, \Sigma, F_1 \cup F_2, \Delta_1 \cup \Delta_2)$

(assumendo $Q_1 \cap Q_2 = \emptyset$)

Chiaramente: $L(A) = L(A_1) \cup L(A_2)$ ■

Dimostrazione (Intersezione - Product Construction):

$$A = (Q_1 \times Q_2, \Sigma, F_1 \times F_2, \Delta)$$

dove:

$$\Delta = \{f((q_1, q_2), \dots, (q_n, q_{n'})) \rightarrow (p_1, p_2) :$$

$$f(q_1, \dots, q_n) \rightarrow p_1 \in \Delta_1 \wedge$$

$$f(q_2, \dots, q_{n'}) \rightarrow p_2 \in \Delta_2\}$$

Dimostrazione (Complemento):

Dato A deterministico completo:

$$A' = (Q, \Sigma, Q \setminus Q_f, \Delta)$$

$$L(A') = T_\Sigma \setminus L(A)$$

Per rendere A completo: aggiungi stato sink $q_\perp$ e transizioni mancanti. ■

2.8.3 Minimizzazione di Tree Automata

*Definizione 2.8.6 (Equivalenza di stati):**

$$p \equiv q \Leftrightarrow \forall \text{ contesto } C[_]: C[p] \in Q_f \Leftrightarrow C[q] \in Q_f$$

*Algoritmo 2.8.1 (Minimizzazione - Hopcroft-style):**

Input: Bottom-up tree automaton A deterministico

Output: Tree automaton minimale $A_{\min}$

1. Inizializza partizione: $P_0 = \{Q_f, Q \setminus Q_f\}$

2. Raffina iterativamente:

$$P_{\{i+1\}} = \text{raffina}(P_i, \Delta)$$

fino a punto fisso

3. Costruisci $A_{\min}$ con stati = classi di equivalenza finali

*Teorema 2.8.2 (Unicità del minimo):**

Il tree automaton deterministico minimale è unico (a meno di isomorfismo)

2.8.4 Top-Down Tree Automata

*Definizione 2.8.7 (Top-Down Tree Automaton):**

$A = (Q, \Sigma, Q_0, \Delta)$

Transizioni: $q \rightarrow f(q_1, \dots, q_n)$

Run: assegnazione di stati procedendo dalla radice alle foglie

*Teorema 2.8.3 (Equivalenza Bottom-Up $\leftrightarrow$ Top-Down):**

$\forall$ linguaggio regolare di alberi L :

L è riconosciuto da bottom-up TA $\Leftrightarrow L$ è riconosciuto da top-down TA

Dimostrazione (Schema):

(BU $\rightarrow$ TD): Dato BU automaton A_{BU} , costruiamo TD automaton A_{TD} che:

- Simula A_{BU} "all'indietro"
- Stati di A_{TD} = potenziali assegnamenti bottom-up
- Transizioni verificano compatibilità

(TD $\rightarrow$ BU): Simile ma inverso. ■

2.8.5 Tree Transducers

*Definizione 2.8.8 (Top-Down Tree Transducer):**

$T = (Q, \Sigma, \Delta, q_0, R)$

Regole R:

$$q(\sigma(x_1, \dots, x_n)) \rightarrow \delta(q_1(x_{\pi(1)}), \dots, q_m(x_{\pi(m)}))$$

dove:

- $\sigma \in \Sigma_n$ (input symbol)
- $\delta \in \Delta_m$ (output symbol)
- π : permutazione dei figli

*Definizione 2.8.9 (Relazione indotta):**

$$\tau(T) = \{(s, t) : s \rightarrow^*_T t\}$$

dove $s \rightarrow^*_T t$ significa:

$\exists$ derivazione da $q_0(s)$ a t usando regole di T

*Teorema 2.8.4 (Composizione di transducers):**

La composizione di k top-down tree transducers è ancora un top-down transducer
(con look-ahead e stato di output)

2.8.6 Applicazioni: XML Processing

Modello 2.8.1 (XML Schema come Tree Automaton):

Un XML Schema definisce:

- Elementi (Σ_n per vari n)
- Struttura nidificata (transizioni del TA)

Validation = Membership checking nel linguaggio del TA

Algoritmo: Bottom-up traversal dell'XML tree

Complessità: $O(|\text{document}|)$

Modello 2.8.2 (XSLT come Tree Transducer):

XSLT transformation $\approx$ Tree Transducer

Template:

```
<xsl:template match="pattern">
  <output>
```

Corrispondenza con regola di transducer:

$q(\text{pattern}) \rightarrow \text{output}[q'(\dots)]$

2.9 Automi a Pila Visibilmente (Visibly Pushdown Automata)

2.9.1 Definizione e Motivazione

*Definizione 2.9.1 (Alfabeto visibilmente partizionato):**

$\Sigma = \Sigma_{\text{call}} \cup \Sigma_{\text{ret}} \cup \Sigma_{\text{int}}$

dove:

- Σ_{call} : simboli che causano push
- Σ_{ret} : simboli che causano pop
- Σ_{int} : simboli interni (no stack operation)

*Definizione 2.9.2 (VPA - Visibly Pushdown Automaton):**

$$V = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{ret}} \cup \Sigma_{\text{int}}, \Gamma, \delta, q_0, F)$$

Transizioni:

- $\delta_{\text{call}}: Q \times \Sigma_{\text{call}} \rightarrow Q \times \Gamma$ (push)
- $\delta_{\text{ret}}: Q \times \Sigma_{\text{ret}} \times \Gamma \rightarrow Q$ (pop)
- $\delta_{\text{int}}: Q \times \Sigma_{\text{int}} \rightarrow Q$ (no stack change)

*Teorema 2.9.1 (Determinizzazione dei VPA):**

$\forall$ VPA non-deterministico N: $\exists$ VPA deterministico D con $L(D) = L(N)$

Dimostrazione: Subset construction come per DFA, ma tracking configurazioni:

Stati di D: $P(Q \times \text{Stack_symbols})$

Deterministico perché stack operations sono determinate dall'input symbol. ■

2.9.2 Proprietà di Chiusura

*Teorema 2.9.2 (Chiusura dei linguaggi VPA):**

I linguaggi VPA sono chiusi sotto:

- Unione
- Intersezione
- Complemento
- Concatenazione
- Kleene star

Nota: Questa è una differenza cruciale rispetto ai CFL generali (non chiusi sotto complemento/intersezione).

2.9.3 Applicazioni

Applicazione 2.9.1 (Parsing di linguaggi con matching brackets):

XML, JSON, linguaggi di programmazione con blocchi

Esempio:

$\Sigma_{\text{call}} = \{ '(', '[', '{' \}$

$\Sigma_{\text{ret}} = \{ ')', ']', '}' \}$

$\Sigma_{\text{int}} = \text{altri caratteri}$

VPA verifica matching corretto in tempo lineare $O(n)$

*Teorema 2.9.3 (Complessità membership per VPA):**

Membership per VPA deterministici: $O(n)$ tempo, $O(n)$ spazio

Membership per VPA non-deterministici: $O(n^3)$ tempo (via CYK-like algorithm)

2.10 Timed Automata

2.10.1 Modellazione del Tempo

*Definizione 2.10.1 (Clock):**

Clock: variabile reale $x \in \mathbb{R}_{\geq 0}$ che aumenta uniformemente

*Definizione 2.10.2 (Clock Constraint):**

$\psi ::= x \sim c \mid \psi \wedge \psi \mid \neg \psi$

dove:

- x è un clock

- $\sim \in \{<, \leq, =, \geq, >\}$

- $c \in \mathbb{Q}_{\geq 0}$

*Definizione 2.10.3 (Timed Automaton):**

$A = (L, L_0, \Sigma, X, E, \text{Inv})$

dove:

- L : locations (stati)
- $L_0 \subseteq L$: initial locations
- Σ : alfabeto di azioni
- X : insieme finito di clocks
- $E \subseteq L \times \Sigma \times 2^X \times \Phi(X) \times L$: transizioni
(source, action, reset, guard, target)
- $\text{Inv}: L \rightarrow \Phi(X)$: invariants per location

2.10.2 Semantica

*Definizione 2.10.4 (Configurazione):**

Configurazione: (l, v)

dove:

- $l \in L$: location corrente
- $v: X \rightarrow \mathbb{R}_{\geq 0}$: valutazione dei clocks

Semantica operativa:

1. Delay transition:

$(l, v) \xrightarrow{\delta} (l, v + \delta)$

se $v + \delta$ soddisfa $\text{Inv}(l)$

2. Action transition:

$$(l, v) \xrightarrow{a} (l', v')$$

se $\exists (l, a, R, g, l') \in E$:

v soddisfa g

$$v' = v[R := 0]$$

v' soddisfa $\text{Inv}(l')$

2.10.3 Raggiungibilità

Problema 2.10.1 (Reachability):

Input: Timed Automaton A , location l_f

Output: Decidere se $\exists$ run da $(l_0, 0)$ a (l_f, v) per qualche v

*Teorema 2.10.1 (Decidibilità della raggiungibilità):**

Il problema di raggiungibilità per timed automata è decidibile

Dimostrazione (Schema):

1. Costruire il region graph (numero finito di regioni)
2. Ogni regione rappresenta un insieme di valutazioni "equivalenti"
3. Raggiungibilità in region graph $\Rightarrow$ raggiungibilità in TA

Complessità: PSPACE-complete ■

2.10.4 Zone Abstraction

*Definizione 2.10.5 (Zone):**

Una zone è un insieme convesso di valutazioni definito da:

Difference Bound Matrix (DBM)

Z : insieme di vincoli $x_i - x_j \sim c_{\{ij\}}$

*Algoritmo 2.10.1 (Forward Reachability con Zones):**

Input: TA A, target location

Output: Raggiungibile?

1. Init: $Z_0 = \{v \mid v(x) = 0 \ \forall x \in X, v \text{ satisfies } \text{Inv}(I_0)\}$

2. Iterate:

- Compute successor zones via actions
- Compute time-successors
- Apply invariants

3. Check if target reached

*Teorema 2.10.2 (Correttezza di zone abstraction):**

Zone abstraction è sound e complete per reachability

Parte II: Espressioni Regolari e Proprietà dei Linguaggi Regolari

Capitolo 3: Espressioni Regolari e Loro Equivalenza con gli Automi

3.1 Definizione Formale delle Espressioni Regolari

Le espressioni regolari costituiscono un formalismo dichiarativo per la descrizione dei linguaggi regolari, complementare alla caratterizzazione operativa fornita dagli automi a stati finiti. La loro importanza risiede nella capacità di specificare linguaggi in modo conciso e composizionale, sfruttando operatori algebrici sulle stringhe.

Definizione 3.1.1 (Espressioni regolari): Sia Σ un alfabeto finito. L'insieme delle espressioni regolari su Σ , denotato con $\text{REG}(\Sigma)$, è definito induttivamente come segue:

- (i) Il simbolo $\emptyset$ è una espressione regolare, che denota il linguaggio vuoto.
- (ii) Il simbolo ε è una espressione regolare, che denota il linguaggio $\{\varepsilon\}$ contenente solo la stringa vuota.
- (iii) Per ogni $a \in \Sigma$, il simbolo a è una espressione regolare, che denota il linguaggio $\{a\}$ contenente solo la stringa di lunghezza uno formata dal simbolo a .

(iv) Se r_1 e r_2 sono espressioni regolari, allora $(r_1 + r_2)$ è una espressione regolare, che denota il linguaggio $L(r_1) \cup L(r_2)$. Questa operazione è chiamata unione o alternativa.

(v) Se r_1 e r_2 sono espressioni regolari, allora $(r_1 r_2)$ è una espressione regolare, che denota il linguaggio $L(r_1)L(r_2)$. Questa operazione è chiamata concatenazione.

(vi) Se r è una espressione regolare, allora (r^*) è una espressione regolare, che denota il linguaggio $L(r)^*$. Questa operazione è chiamata chiusura di Kleene o stella di Kleene.

(vii) Nessun'altra stringa è una espressione regolare se non può essere ottenuta mediante l'applicazione finita delle regole precedenti.

La funzione $L : \text{REG}(\Sigma) \rightarrow \mathcal{P}(\Sigma^*)$ che associa a ciascuna espressione regolare il linguaggio da essa denotato è definita ricorsivamente in accordo con le regole sopra specificate.

Convenzioni Notazionali: Per alleggerire la notazione, adottiamo le seguenti convenzioni:

(i) Le parentesi esterne possono essere omesse quando non vi è ambiguità.

(ii) La stella di Kleene ha la precedenza più alta, seguita dalla concatenazione, e infine dall'unione ha la precedenza più bassa. Così, $ab^* + c$ denota $((a(b^*))) + c$.

(iii) La concatenazione può essere indicata anche mediante giustapposizione senza alcun operatore esplicito.

(iv) Per espressioni regolari r , definiamo r^+ come abbreviazione di rr^* , così che $L(r^+) = L(r)^+$.

(v) Per $n \geq 0$, definiamo r^n induttivamente come: $r^0 = \varepsilon$, $r^{n+1} = rr^n$.

Esempio 3.1.1: Consideriamo l'alfabeto $\Sigma = \{0, 1\}$. Allora:

(a) L'espressione regolare $(0 + 1)^*$ denota il linguaggio Σ^* , ovvero l'insieme di tutte le stringhe binarie.

(b) L'espressione regolare 010 denota il linguaggio di tutte le stringhe binarie contenenti esattamente un simbolo 1.

(c) L'espressione regolare $(0 + 1)^*1$ denota il linguaggio di tutte le stringhe binarie terminanti con il simbolo 1.

(d) L'espressione regolare $(00 + 11)^*$ denota il linguaggio di tutte le stringhe binarie di lunghezza pari formate da blocchi di due simboli identici consecutivi.

(e) L'espressione regolare $(0 + 1)01(0 + 1)$ denota il linguaggio di tutte le stringhe binarie contenenti la sottostringa 01.

3.2 Algebra delle Espressioni Regolari

Le espressioni regolari soddisfano numerose identità algebriche che riflettono le proprietà dei linguaggi che denotano. Formalizziamo questo aspetto introducendo una relazione di equivalenza.

Lemma 3.2.2 (Lemma di sostituzione di Arden generalizzato): Sia $X = \alpha X + \beta$ un'equazione in una variabile X , dove α e β sono espressioni regolari e $\epsilon \notin L(\alpha)$. Allora $X = \alpha^* \beta$ è l'unica soluzione.

Dimostrazione: Questa è una riaffermazione del Teorema 1.2.3 in termini di espressioni regolari. La dimostrazione è identica, sostituendo linguaggi L con $L(\alpha)$ e M con $L(\beta)$. ■

3.3 Equivalenza tra Espressioni Regolari e Automi

Stabiliamo ora il risultato fondamentale che caratterizza completamente i linguaggi regolari: un linguaggio è regolare se e solo se può essere descritto mediante un'espressione regolare.

Teorema 3.3.1 (Teorema di Kleene): Un linguaggio $L \subseteq \Sigma^*$ è regolare se e solo se esiste un'espressione regolare r tale che $L = L(r)$.

Dimostrazione: La dimostrazione procede in due direzioni.

Direzione ($\Leftarrow$): Ogni linguaggio denotato da un'espressione regolare è regolare.

Procediamo per induzione strutturale sulla definizione di espressione regolare.

Base:

- $L(\emptyset) = \emptyset$ è riconosciuto dal DFA $A_\emptyset = (\{q_0\}, \Sigma, \delta, q_0, \emptyset)$ dove $\delta(q_0, a) = q_0$ per ogni $a \in \Sigma$.
- $L(\epsilon) = \{\epsilon\}$ è riconosciuto dal DFA $A_\epsilon = (\{q_0\}, \Sigma, \delta, q_0, \{q_0\})$ con $\delta(q_0, a) = q_1$ per un nuovo stato non accettante q_1 , e $\delta(q_1, a) = q_1$.
- Per ogni $a \in \Sigma$, $L(a) = \{a\}$ è riconosciuto da un DFA con due stati: uno iniziale non accettante e uno finale raggiungibile solo tramite la transizione etichettata con a .

Passo induttivo:

- Se $L(r_1)$ e $L(r_2)$ sono regolari, allora $L(r_1 + r_2) = L(r_1) \cup L(r_2)$ è regolare per il Teorema 2.1.1.
- Se $L(r_1)$ e $L(r_2)$ sono regolari, allora $L(r_1 r_2) = L(r_1)L(r_2)$ è regolare per il Teorema 2.2.3.
- Se $L(r)$ è regolare, allora $L(r^*) = L(r)^*$ è regolare per il Teorema 2.2.4.

Direzione ($\Rightarrow$): Ogni linguaggio regolare può essere descritto da un'espressione regolare.

Sia L riconosciuto da un DFA $A = (Q, \Sigma, \delta, q_0, F)$ con $Q = \{q_1, q_2, \dots, q_n\}$. Definiamo per ogni $i, j \in \{1, \dots, n\}$ e $k \in \{0, 1, \dots, n\}$:

$$R^k(i, j) = \{w \in \Sigma^* : \delta^*(q_i, w) = q_j \text{ e tutti gli stati intermedi nel calcolo appartengono a } \{q_1, \dots, q_n\}\}$$

Intuitivamente, $R^k(i, j)$ è l'insieme delle stringhe che portano da q_i a q_j passando solo attraverso stati con indice al massimo k come stati intermedi (gli stati q_i e q_j sono esclusi da questa restrizione).

Procediamo per induzione su k per mostrare che ogni $R^k(k)_i$ è rappresentabile mediante un'espressione regolare.

Base ($k = 0$): $R^0(0)_i$ contiene le stringhe che vanno da q_i a q senza passare per alcuno stato intermedio. Quindi:

$$R^0(0)_i = \{a \in \Sigma : \delta(q_i, a) = q\} \cup (\text{se } i = j \text{ allora } \{\epsilon\} \text{ altrimenti } \emptyset)$$

Questo può essere espresso come $r^0(0)_i$ dove:

$$r^0(0)_i = (a_1 + a_2 + \dots + a_n) + (\epsilon \text{ se } i = j, \text{ altrimenti assente})$$

dove $\{a_1, \dots, a_n\} = \{a \in \Sigma : \delta(q_i, a) = q\}$.

Passo induttivo (da $k-1$ a k): Supponiamo di aver costruito $r^{k-1}(k-1)_i$ per tutti i, j . Per costruire $R^k(k)_i$, distinguiamo due casi:

1. I cammini da q_i a q non passano mai per q come stato intermedio. Questi cammini appartengono a $R^{k-1}(k-1)_i$.
2. I cammini da q_i a q passano per q almeno una volta come stato intermedio. Tali cammini possono essere decomposti come:
 - Un cammino da q_i a q senza passare per q (appartenente a $R^{k-1}(k-1)_i$)
 - Seguito da zero o più cammini da q a q senza passare per q (ciascuno in $R^{k-1}(k-1)_{qq}$)
 - Seguito da un cammino da q a q senza passare per q (appartenente a $R^{k-1}(k-1)_{qq}$)

Quindi:

$$R^k(k)_i = R^{k-1}(k-1)_i \cup R^{k-1}(k-1)_i (R^{k-1}(k-1)_{qq})^* R^{k-1}(k-1)_{qq}$$

E possiamo definire:

$$r^k(k)_i = r^{k-1}(k-1)_i + r^{k-1}(k-1)_i (r^{k-1}(k-1)_{qq})^* r^{k-1}(k-1)_{qq}$$

Infine, il linguaggio $L(A)$ è l'unione di tutti i linguaggi $R^n(n)_0$ dove q_0 è lo stato iniziale (supponiamo sia q_1) e $q \in F$:

$$L(A) = \bigcup \{R^n(n)_i : q_i \in F\}$$

Quindi $L(A) = L(r)$ dove:

$$r = \sum \{r^n(n)_i : q_i \in F\}$$

(la somma denota l'unione delle espressioni regolari). ■

Esempio 3.3.1: Consideriamo il DFA A con $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, stato iniziale q_1 , stato finale q_2 , e transizioni:

- $\delta(q_1, a) = q_1$
- $\delta(q_1, b) = q_2$
- $\delta(q_2, a) = q_2$
- $\delta(q_2, b) = q_1$

Calcoliamo le espressioni regolari $R^k(k)_i$:

$k = 0$:

- $r^{\wedge}(0)_{11} = a + \varepsilon$ (da q_1 a q_1 con a , o rimanendo fermi)
- $r^{\wedge}(0)_{12} = b$ (da q_1 a q_2 con b)
- $r^{\wedge}(0)_{21} = b$ (da q_2 a q_1 con b)
- $r^{\wedge}(0)_{22} = a + \varepsilon$ (da q_2 a q_2 con a , o rimanendo fermi)

$k = 1$:

- $r^{\wedge}(1)_{11} = r^{\wedge}(0)_{11} + r^{\wedge}(0)_{11}(r^{\wedge}(0)_{11})^*r^{\wedge}(0)_{11} = (a + \varepsilon) + (a + \varepsilon)(a + \varepsilon)(a + \varepsilon) = (a + \varepsilon)(a + \varepsilon)^* = a^*$
- $r^{\wedge}(1)_{12} = r^{\wedge}(0)_{12} + r^{\wedge}(0)_{11}(r^{\wedge}(0)_{11})^*r^{\wedge}(0)_{12} = b + (a + \varepsilon)(a + \varepsilon)b = ab$
- $r^{\wedge}(1)_{21} = r^{\wedge}(0)_{21} + r^{\wedge}(0)_{21}(r^{\wedge}(0)_{11})^*r^{\wedge}(0)_{11} = b + b(a + \varepsilon)(a + \varepsilon) = ba^*$
- $r^{\wedge}(1)_{22} = r^{\wedge}(0)_{22} + r^{\wedge}(0)_{21}(r^{\wedge}(0)_{11})^*r^{\wedge}(0)_{12} = (a + \varepsilon) + b(a + \varepsilon)b = a + \varepsilon + bab$

$k = 2$:

- $r^{\wedge}(2)_{12} = r^{\wedge}(1)_{12} + r^{\wedge}(1)_{12}(r^{\wedge}(1)_{22})^*r^{\wedge}(1)_{22} = ab + ab(a + \varepsilon + bab)(a + \varepsilon + bab)$

Semplificando: $r^{\wedge}(2)_{12} = ab(a + bab)^*$

Quindi $L(A) = L(ab(a + bab)^*)$.

3.4 Costruzione di Thompson per la Conversione da Espressioni Regolari a NFA

La costruzione di Thompson fornisce un algoritmo efficiente per convertire un'espressione regolare in un NFA con ε -transizioni che riconosce lo stesso linguaggio.

Algoritmo 3.4.1 (Costruzione di Thompson): Data un'espressione regolare r , costruiamo induttivamente un ε -NFA $N(r) = (Q, \Sigma, \Delta, q_0, \{q_f\})$ con esattamente uno stato iniziale e uno stato finale, e tale che $L(N(r)) = L(r)$.

Base:

1. Per $r = \emptyset$: $N(\emptyset)$ ha due stati q_0 e q_f , nessuna transizione. Quindi $L(N(\emptyset)) = \emptyset$.
2. Per $r = \varepsilon$: $N(\varepsilon)$ ha due stati q_0 e q_f , con una ε -transizione da q_0 a q_f . Quindi $L(N(\varepsilon)) = \{\varepsilon\}$.
3. Per $r = a$ con $a \in \Sigma$: $N(a)$ ha due stati q_0 e q_f , con una transizione etichettata a da q_0 a q_f . Quindi $L(N(a)) = \{a\}$.

Passo induttivo:

1. Per $r = r_1 + r_2$: Siano $N(r_1) = (Q_1, \Sigma, \Delta_1, q_0^1, \{qf^1\})$ e $N(r_2) = (Q_2, \Sigma, \Delta_2, q_0^2, \{qf^2\})$ gli ε -NFA per r_1 e r_2 (assumiamo $Q_1 \cap Q_2 = \emptyset$). Costruiamo $N(r_1 + r_2)$ come segue:
 - Introduciamo due nuovi stati q_0 e qf
 - $Q = Q_1 \cup Q_2 \cup \{q_0, qf\}$
 - Aggiungiamo ε -transizioni: $q_0 \rightarrow q_0^1$, $q_0 \rightarrow q_0^2$, $qf^1 \rightarrow qf$, $qf^2 \rightarrow qf$
 - Manteniamo tutte le transizioni di $N(r_1)$ e $N(r_2)$
2. Intuitivamente, l'automa sceglie non-deterministicamente di simulare $N(r_1)$ o $N(r_2)$.
3. Per $r = r_1 r_2$: Con le stesse convenzioni, costruiamo:
 - $Q = Q_1 \cup Q_2 \cup \{q_0, qf\}$
 - Aggiungiamo ε -transizioni: $q_0 \rightarrow q_0^1$, $qf^1 \rightarrow q_0^2$, $qf^2 \rightarrow qf$
 - Manteniamo tutte le transizioni di $N(r_1)$ e $N(r_2)$
4. L'automa prima simula $N(r_1)$, poi passa a $N(r_2)$.
5. Per $r = r_1^*$: Costruiamo:
 - $Q = Q_1 \cup \{q_0, qf\}$
 - Aggiungiamo ε -transizioni: $q_0 \rightarrow q_0^1$, $q_0 \rightarrow qf$, $qf^1 \rightarrow q_0^1$, $qf^1 \rightarrow qf$
 - Manteniamo tutte le transizioni di $N(r_1)$
6. L'automa può scegliere di accettare immediatamente (ε), o di simulare $N(r_1)$ ripetutamente.

Teorema 3.4.1 (Correttezza della costruzione di Thompson): Per ogni espressione regolare r , $L(N(r)) = L(r)$.

Dimostrazione: Per induzione strutturale su r . I casi base sono immediati. Per i casi induttivi, utilizziamo la semantica delle operazioni sugli NFA dimostrata nei teoremi 2.2.3 e 2.2.4, e la chiusura sotto unione (che si dimostra analogamente). ■

Teorema 3.4.2 (Complessità della costruzione di Thompson): Data un'espressione regolare r di lunghezza m (numero di simboli), la costruzione di Thompson produce un ε -NFA con al massimo $2m$ stati e al massimo $4m$ transizioni.

Dimostrazione: Ogni operatore nell'espressione regolare introduce al massimo 2 nuovi stati e al massimo 4 nuove transizioni (come si può verificare dall'algoritmo). Poiché ci sono al massimo m operatori, il risultato segue. ■

3.5 Algoritmi per Espressioni Regolari

Definizione 3.5.1 (Derivata di Brzowski): La derivata di un'espressione regolare r rispetto a un simbolo a , denotata $\partial_a(r)$, è definita induttivamente:

- $\partial_a(\emptyset) = \emptyset$
- $\partial_a(\varepsilon) = \emptyset$

- $\partial_a(b) = \varepsilon$ se $a = b$, $\emptyset$ altrimenti
- $\partial_a(r_1 + r_2) = \partial_a(r_1) + \partial_a(r_2)$
- $\partial_a(r_1 r_2) = \partial_a(r_1) r_2 + v(r_1) \partial_a(r_2)$
- $\partial_a(r^*) = \partial_a(r) r^*$

dove $v(r) = \varepsilon$ se $\varepsilon \in L(r)$, $\emptyset$ altrimenti.

Teorema 3.5.1 (Correttezza della derivata): $L(\partial_a(r)) = \{w : aw \in L(r)\}$

Teorema 3.5.2 (Finitezza delle derivate): Per ogni espressione regolare r , l'insieme $\{\partial_w(r) : w \in \Sigma^*\}$ modulo equivalenza contiene solo un numero finito di elementi.

Algoritmo 3.5.1 (Costruzione DFA tramite derivate):

DFA_from_regex(r):

$Q := \{r\}$

$W := \{r\}$ // worklist

$\delta := \emptyset$

while $W \neq \emptyset$:

$r' := W.pop()$

 for $a \in \Sigma$:

$r'' := \text{simplify}(\partial_a(r'))$

 if $r'' \notin Q$:

$Q := Q \cup \{r''\}$

$W := W \cup \{r''\}$

$\delta := \delta \cup \{(r', a, r'')\}$

$F := \{r' \in Q : v(r') = \varepsilon\}$

return $(Q, \Sigma, \delta, r, F)$

Complessità: $O(2^n)$ nel caso peggiore dove n è la dimensione della regex.

3.6 Espressioni Regolari Estese

Definizione 3.6.1 (Extended Regular Expressions): Le espressioni regolari estese aggiungono operatori:

- Complemento: $\neg r$ con $L(\neg r) = \Sigma^* \setminus L(r)$
- Intersezione: $r_1 \wedge r_2$ con $L(r_1 \wedge r_2) = L(r_1) \cap L(r_2)$
- Differenza: $r_1 \setminus r_2$ con $L(r_1 \setminus r_2) = L(r_1) \setminus L(r_2)$

Teorema 3.6.1 (Potere espressivo invariato): Le espressioni regolari estese generano esattamente i linguaggi regolari.

Dimostrazione: I linguaggi regolari sono chiusi sotto complemento (Teorema 2.1.3) e intersezione (Teorema 2.1.2). La differenza si esprime come $r_1 \setminus r_2 \equiv r_1 \wedge \neg r_2$. ■

Teorema 3.6.2 (Blow-up esponenziale): Convertire un'espressione regolare estesa con complemento in una espressione regolare standard può richiedere un aumento esponenziale nella dimensione.

Capitolo 4: Lemma di Pumping e Proprietà di Chiusura

4.1 Il Lemma di Pumping per i Linguaggi Regolari

Il lemma di pumping fornisce una condizione necessaria affinché un linguaggio sia regolare. Esso costituisce uno strumento fondamentale per dimostrare che certi linguaggi non sono regolari.

Teorema 4.1.1 (Lemma di Pumping per linguaggi regolari): Sia L un linguaggio regolare. Allora esiste una costante positiva p (la "lunghezza di pumping") tale che ogni stringa $w \in L$ con $|w| \geq p$ può essere decomposta come $w = xyz$, dove:

(i) $|xy| \leq p$ (ii) $|y| \geq 1$ (iii) Per ogni $i \geq 0$, $xy^i z \in L$

Dimostrazione: Sia $A = (Q, \Sigma, \delta, q_0, F)$ un DFA che riconosce L , e sia $p = |Q|$ il numero di stati di A . Consideriamo una stringa $w \in L$ con $|w| \geq p$, diciamo $w = a_1 a_2 \dots a_n$ con $n \geq p$.

Il calcolo di A su w produce una sequenza di $p+1$ stati:

$$q_0, \delta(q_0, a_1), \delta^*(q_0, a_1 a_2), \dots, \delta^*(q_0, a_1 \dots a_n)$$

Poiché A ha solo p stati distinti, per il principio del cassetto (pigeonhole principle) devono esistere due indici $0 \leq i < j \leq p$ tali che:

$$\delta^*(q_0, a_1 \dots a_i) = \delta^*(q_0, a_1 \dots a_j)$$

Definiamo:

- $x = a_1 \dots a_i$
- $y = a_{i+1} \dots a_j$
- $z = a_{j+1} \dots a_n$

Allora $w = xyz$, e per costruzione:

- $|xy| = j \leq p$ (quindi (i) è soddisfatta)
- $|y| = j - i \geq 1$ (quindi (ii) è soddisfatta)

Per verificare (iii), osserviamo che:

$$\delta^*(q_0, x) = \delta^*(q_0, xy) = \delta^*(q_0, a_1 \dots a_j) = q \text{ (per qualche stato } q)$$

$$\text{Inoltre: } \delta^*(q, y) = \delta^*(\delta^*(q_0, x), y) = \delta^*(q_0, xy) = q$$

Quindi y rappresenta un ciclo dallo stato q a se stesso. Per ogni $i \geq 0$:

$$\delta^*(q_0, xy^i z) = \delta^*(\delta^*(q, y^i), z) = \delta^*(q, z) = \delta^*(q_0, xyz) \in F$$

Quindi $xy^i z \in L$ per ogni $i \geq 0$. ■

Corollario 4.1.2: Il lemma di pumping può essere riformulato contrapositivamente: se un linguaggio L non soddisfa la conclusione del lemma per alcuna scelta di p , allora L non è regolare.

Esempio 4.1.1: Dimostriamo che il linguaggio $L = \{a^n b^n : n \geq 0\}$ non è regolare.

Dimostrazione: Procediamo per contraddizione. Supponiamo che L sia regolare e sia p la costante di pumping garantita dal lemma. Consideriamo la stringa $w = a^p b^p \in L$. Poiché $|w| = 2p \geq p$, il lemma di pumping garantisce che w può essere scritta come $w = xyz$ dove $|xy| \leq p$, $|y| \geq 1$, e $xy^i z \in L$ per ogni $i \geq 0$.

Poiché $|xy| \leq p$, la sottostringa xy è contenuta interamente nella parte iniziale a^p . Quindi $y = a^k$ per qualche $k \geq 1$. Consideriamo il caso $i = 2$:

$$xy^2z = a^p b^p \text{ con } y \text{ ripetuto due volte} = a^{(p+k)} b^p$$

Ma $p + k > p$, quindi $xy^2z \notin L$, contraddicendo il lemma di pumping. Quindi L non può essere regolare. ■

Esempio 4.1.2: Il linguaggio $L = \{w \in \{0, 1\}^* : w \text{ contiene lo stesso numero di 0 e di 1}\}$ non è regolare.

Dimostrazione: Supponiamo per contraddizione che L sia regolare con costante di pumping p . Consideriamo $w = 0^p 1^p \in L$. Per il lemma di pumping, $w = xyz$ con $|xy| \leq p$, $|y| \geq 1$, e $xy^i z \in L$ per ogni $i \geq 0$.

Poiché $|xy| \leq p$, abbiamo $y = 0^k$ con $k \geq 1$. Scegliendo $i = 0$, otteniamo:

$$xz = 0^{(p-k)} 1^p$$

che non appartiene a L poiché contiene $p - k < p$ zeri e p uni. Contraddizione. ■

Esempio 4.1.3: Il linguaggio $L = \{a^{(n^2)} : n \geq 0\}$ non è regolare.

Dimostrazione: Sia p la costante di pumping. Consideriamo $w = a^{(p^2)} \in L$. Scriviamo $w = xyz$ con $|xy| \leq p$, $|y| = k \geq 1$. Allora $|xy^2z| = p^2 + k$.

Affinché $xy^2z \in L$, dovrebbe esistere m tale che $p^2 + k = m^2$. Ma:

$$p^2 < p^2 + k \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$$

Quindi $p^2 < m^2 < (p+1)^2$, il che implica $p < m < p+1$. Ma ciò è impossibile per m intero. Contraddizione. ■

4.2 Versione Forte del Lemma di Pumping

Esiste una versione rafforzata del lemma di pumping che fornisce maggiore flessibilità nelle dimostrazioni di non-regolarità.

Teorema 4.2.1 (Lemma di Pumping forte): Sia L un linguaggio regolare. Allora esiste una costante p tale che per ogni stringa $w \in L$ con $|w| \geq p$, esistono stringhe x, y, z con $w = xyz$ tali che:

(i) $|y| \geq 1$ (ii) $|xy| \leq p$ (iii) Per ogni $i \geq 0$, $xy^i z \in L$

Inoltre, esistono infiniti indici i per cui $xy^i z$ ha una decomposizione simile.

Questa versione è equivalente a quella standard ma enfatizza che il "pompaggio" può essere iterato indefinitamente.

4.3 Proprietà di Chiusura dei Linguaggi Regolari

I linguaggi regolari godono di numerose proprietà di chiusura rispetto alle operazioni insiemistiche e stringali.

Teorema 4.3.1 (Chiusura rispetto alle operazioni booleane): La classe dei linguaggi regolari è chiusa rispetto alle operazioni di unione, intersezione, complemento e differenza.

Dimostrazione: Unione, intersezione e complemento sono stati dimostrati nei Teoremi 2.1.1, 2.1.2, 2.1.3. Per la differenza, osserviamo che $L_1 \setminus L_2 = L_1 \cap \bar{L}_2$, quindi la chiusura segue dalla chiusura sotto intersezione e complemento. ■

Teorema 4.3.2 (Chiusura rispetto alla concatenazione): Se L_1 e L_2 sono linguaggi regolari, allora $L_1 L_2$ è regolare.

Dimostrazione: Già dimostrato nel Teorema 2.2.3. ■

Teorema 4.3.3 (Chiusura rispetto alla chiusura di Kleene): Se L è un linguaggio regolare, allora L^* è regolare.

Dimostrazione: Già dimostrato nel Teorema 2.2.4. ■

Definizione 4.3.1 (Inversione di un linguaggio): Sia $L \subseteq \Sigma^*$ un linguaggio. L'inversione di L , denotata L^R , è definita come:

$$L^R = \{w^R : w \in L\}$$

dove w^R è l'inversione della stringa w definita nella Definizione 1.1.5.

Teorema 4.3.4 (Chiusura rispetto all'inversione): Se L è un linguaggio regolare, allora L^R è regolare.

Dimostrazione: Sia $A = (Q, \Sigma, \delta, q_0, F)$ un NFA che riconosce L . Costruiamo un NFA $A^R = (Q, \Sigma, \Delta^R, F, \{q_0\})$ che riconosce L^R come segue:

- Gli stati rimangono gli stessi
- Gli stati iniziali di A^R sono gli stati finali di A
- L'unico stato finale di A^R è lo stato iniziale di A
- Le transizioni sono invertite: $a \in \Delta^R(q_2, a)$ se e solo se $q_2 \in \Delta(q_1, a)$

Intuitivamente, A^R simula A all'indietro. Una stringa w è accettata da A^R se e solo se w^R è accettata da A , quindi $L(A^R) = L(A)^R = L^R$. ■

Definizione 4.3.2 (Omomorfismo di stringhe): Siano Σ_1 e Σ_2 alfabeti. Una funzione $h : \Sigma_1^* \rightarrow \Sigma_2^*$ è un omomorfismo se:

(i) $h(\varepsilon) = \varepsilon$ (ii) $h(w_1 w_2) = h(w_1) h(w_2)$ per ogni $w_1, w_2 \in \Sigma_1^*$

Un omomorfismo è completamente determinato dalla sua azione sui simboli dell'alfabeto, ovvero dai valori $h(a)$ per $a \in \Sigma_1$. Per ogni stringa $w = a_1 a_2 \dots a_n$, vale:

$$h(w) = h(a_1) h(a_2) \dots h(a_n)$$

Definizione 4.3.3 (Immagine diretta e inversa): Sia $h : \Sigma_1^* \rightarrow \Sigma_2^*$ un omomorfismo e $L \subseteq \Sigma_1^*$ un linguaggio.

- L'immagine diretta di L tramite h è: $h(L) = \{h(w) : w \in L\}$
- L'immagine inversa di $K \subseteq \Sigma_2^*$ tramite h è: $h^{-1}(K) = \{w \in \Sigma_1^* : h(w) \in K\}$

Teorema 4.3.5 (Chiusura rispetto agli omomorfismi): Se L è un linguaggio regolare e h è un omomorfismo, allora $h(L)$ è regolare.

Dimostrazione: Sia r un'espressione regolare tale che $L = L(r)$. Definiamo induttivamente un'espressione regolare $h(r)$ sostituendo ogni simbolo a in r con $h(a)$:

- $h(\varnothing) = \varnothing$
- $h(\varepsilon) = \varepsilon$
- $h(a) = \text{espressione regolare che denota } \{h(a)\}$
- $h(r_1 + r_2) = h(r_1) + h(r_2)$
- $h(r_1 r_2) = h(r_1) h(r_2)$
- $h(r^*) = h(r)^*$

Si dimostra per induzione che $L(h(r)) = h(L(r))$. Quindi $h(L) = L(h(r))$ è regolare. ■

Teorema 4.3.6 (Chiusura rispetto all'immagine inversa): Se K è un linguaggio regolare e h è un omomorfismo, allora $h^{-1}(K)$ è regolare.

Dimostrazione: Sia $A = (Q, \Sigma_2, \delta, q_0, F)$ un DFA che riconosce K . Costruiamo un DFA $A' = (Q, \Sigma_1, \delta', q_0, F)$ che riconosce $h^{-1}(K)$ definendo:

$$\delta'(q, a) = \delta^*(q, h(a))$$

dove δ^* è la funzione di transizione estesa di A . Intuitivamente, A' simula A processando $h(a)$ quando legge il simbolo a .

Per induzione su $|w|$, si dimostra che $\delta'(q_0, w) = \delta(q_0, h(w))$. Quindi:

$$w \in L(A') \Leftrightarrow \delta'(q_0, w) \in F \Leftrightarrow \delta(q_0, h(w)) \in F \Leftrightarrow h(w) \in K \Leftrightarrow w \in h^{-1}(K)$$

Quindi $L(A') = h^{-1}(K)$. ■

Definizione 4.3.4 (Quoziente di linguaggi): Siano $L_1, L_2 \subseteq \Sigma^*$ linguaggi. Il quoziente sinistro di L_1 rispetto a L_2 è:

$$L_1 / L_2 = \{w \in \Sigma^* : \text{esiste } v \in L_2 \text{ tale che } wv \in L_1\}$$

Analogamente, il quoziente destro è:

$$L_2 \setminus L_1 = \{w \in \Sigma^* : \text{esiste } v \in L_2 \text{ tale che } vw \in L_1\}$$

Teorema 4.3.7 (Chiusura rispetto al quoziente): Se L_1 è un linguaggio regolare, allora L_1 / L_2 è regolare per ogni linguaggio L_2 .

Dimostrazione: Sia $A = (Q, \Sigma, \delta, q_0, F)$ un DFA che riconosce L_1 . Definiamo un nuovo insieme di stati finali:

$$F' = \{q \in Q : \text{esiste } v \in L_2 \text{ tale che } \delta^*(q, v) \in F\}$$

Sia $A' = (Q, \Sigma, \delta, q_0, F')$. Allora:

$$w \in L(A') \Leftrightarrow \delta^*(q_0, w) \in F' \Leftrightarrow \text{esiste } v \in L_2 \text{ tale che } \delta^*(\delta^*(q_0, w), v) \in F \Leftrightarrow \text{esiste } v \in L_2 \text{ tale che } \delta^*(q_0, wv) \in F \Leftrightarrow \text{esiste } v \in L_2 \text{ tale che } wv \in L_1 \Leftrightarrow w \in L_1 / L_2$$

Quindi $L(A') = L_1 / L_2$, e L_1 / L_2 è regolare. ■

Definizione 4.3.5 (Sostituzione): Una sostituzione è una funzione $s : \Sigma \rightarrow \mathcal{P}(\Gamma^*)$ che associa a ciascun simbolo di Σ un linguaggio su Γ . La sostituzione si estende a stringhe e linguaggi:

- $s(\varepsilon) = \{\varepsilon\}$
- $s(wa) = s(w)s(a)$ per $w \in \Sigma^*, a \in \Sigma$
- $s(L) = \bigcup \{s(w) : w \in L\}$

Teorema 4.3.8 (Chiusura rispetto alle sostituzioni regolari): Se L è un linguaggio regolare su Σ e s è una sostituzione tale che $s(a)$ è regolare per ogni $a \in \Sigma$, allora $s(L)$ è regolare.

Dimostrazione: Sia r un'espressione regolare per L . Per ogni $a \in \Sigma$, sia r_a un'espressione regolare per $s(a)$. Definiamo $s(r)$ sostituendo ogni occorrenza di a in r con r_a :

- $s(\varnothing) = \varnothing$
- $s(\varepsilon) = \varepsilon$
- $s(a) = r_a$
- $s(r_1 + r_2) = s(r_1) + s(r_2)$
- $s(r_1 r_2) = s(r_1) s(r_2)$
- $s(r^*) = s(r)^*$

Si dimostra per induzione che $L(s(r)) = s(L(r))$. Quindi $s(L) = L(s(r))$ è regolare. ■

Teorema 4.3.9 (Non chiusura rispetto al complemento infinito): I linguaggi regolari non sono chiusi rispetto all'operazione di complemento infinito, definita come $\Sigma^* \setminus L$ quando Σ non è l'alfabeto originale ma un'estensione infinita.

Questo teorema mostra che alcune operazioni che sembrano naturali non preservano la regolarità.

4.4 Applicazioni Avanzate del Lemma di Pumping

Esempio 4.4.1 (Linguaggio dei numeri primi in unario): $L = \{1^p : p \text{ è primo}\}$

Dimostrazione che L non è regolare: Supponiamo L regolare con costante di pumping p . Consideriamo $w = 1^q$ dove $q > p$ è primo. Per il lemma, $w = xyz$ con $|xy| \leq p$, $|y| = k \geq 1$, e $xy^i z \in L$ per ogni $i \geq 0$.

Scegliamo $i = q + 1$. Allora: $|xy^{q+1}z| = |xyz| + qk = q + qk = q(1 + k)$

Poiché $1 < 1+k \leq p+1$ e q è primo con $q > p \geq p+1$, abbiamo che $q(1+k)$ è composto (prodotto di due fattori > 1). Contraddizione. ■

Teorema 4.4.1 (Pumping Lemma per lunghezze): Se L è regolare, allora esiste N tale che: $\forall n > N: (|L \cap \Sigma^n| > 0 \Rightarrow |L \cap \Sigma^{n+p}| > 0)$

dove p è la costante di pumping.

4.5 Teoremi di Chiusura Aggiuntivi

Teorema 4.5.1 (Chiusura sotto shuffle): Se L_1, L_2 sono regolari, allora anche: $L_1 \sqcup L_2 = \{w : w = u_1 v_1 u_2 v_2 \dots u_n v_n, u_1 \dots u_n \in L_1, v_1 \dots v_n \in L_2\}$ è regolare.

Dimostrazione: Costruiamo un NFA che simula non-deterministicamente un DFA per L_1 e uno per L_2 in parallelo, alternando tra loro ad ogni simbolo. ■

Teorema 4.5.2 (Chiusura sotto commutazione ciclica): Se L è regolare, allora: $\text{CYCLE}(L) = \{vu : uv \in L\}$ è regolare.

Teorema 4.5.3 (Chiusura sotto quoziente ciclico): Se L è regolare, allora: $L / L = \{w : \exists v \in L \text{ tale che } wv \in L\}$ è regolare (già dimostrato come Teorema 4.3.7).

Parte III: Gerarchia di Chomsky e Grammatiche Context-Free

Capitolo 5: Grammatiche Formali e la Gerarchia di Chomsky

5.1 Grammatiche Generative

Le grammatiche formali forniscono un meccanismo generativo per descrivere linguaggi, complementare alla caratterizzazione riconoscitiva fornita dagli automi. Il lavoro pionieristico di Noam Chomsky negli anni Cinquanta stabilì una gerarchia di classi di grammatiche e linguaggi che è diventata fondamentale sia nella teoria dei linguaggi formali che nella linguistica computazionale.

Definizione 5.1.1 (Grammatica formale): Una grammatica formale è una quadrupla $G = (V, \Sigma, P, S)$ dove:

(i) V è un insieme finito di simboli non terminali (o variabili) (ii) Σ è un insieme finito di simboli terminali (l'alfabeto), con $V \cap \Sigma = \emptyset$ (iii) $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^+$ è un insieme finito di produzioni (o regole di riscrittura) (iv) $S \in V$ è il simbolo iniziale (o assioma)

Le produzioni sono generalmente scritte nella forma $\alpha \rightarrow \beta$ dove $(\alpha, \beta) \in P$. Denotiamo con $(V \cup \Sigma)^*$ l'insieme di tutte le stringhe su $V \cup \Sigma$, e con $(V \cup \Sigma)^+$ l'insieme delle stringhe non vuote.

Definizione 5.1.2 (Derivazione): Sia $G = (V, \Sigma, P, S)$ una grammatica. Diciamo che una stringa γ deriva direttamente una stringa δ , scritto $\gamma \Rightarrow_G \delta$ (o semplicemente $\gamma \Rightarrow \delta$ quando G è chiara dal contesto), se esistono stringhe $\alpha, \beta, \xi, \eta \in (V \cup \Sigma)^*$ tali che:

- $\gamma = \xi\alpha\eta$
- $\delta = \xi\beta\eta$
- $\alpha \rightarrow \beta \in P$

Intuitivamente, δ si ottiene da γ sostituendo una occorrenza di α con β usando una produzione di P .

La chiusura riflessiva e transitiva di $\Rightarrow$, denotata $\Rightarrow^*$, rappresenta una derivazione in zero o più passi:

- $\gamma \Rightarrow^0 \gamma$ per ogni γ (riflessività)
- Se $\gamma \Rightarrow^i \delta$ e $\delta \Rightarrow \eta$, allora $\gamma \Rightarrow^{i+1} \eta$

Scriviamo $\gamma \Rightarrow^* \delta$ per indicare una derivazione in uno o più passi.

Definizione 5.1.3 (Linguaggio generato): Il linguaggio generato da una grammatica $G = (V, \Sigma, P, S)$ è:

$$L(G) = \{w \in \Sigma^* : S \Rightarrow^* w\}$$

Ovvero, $L(G)$ consiste di tutte le stringhe di simboli terminali derivabili dal simbolo iniziale.

Esempio 5.1.1: Consideriamo la grammatica $G = (\{S\}, \{a, b\}, P, S)$ dove P contiene le produzioni:

- $S \rightarrow aSb$
- $S \rightarrow \varepsilon$

Deriviamo la stringa a^2b^2 :

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

In generale, questa grammatica genera il linguaggio $L(G) = \{a^n b^n : n \geq 0\}$, che abbiamo dimostrato non essere regolare nell'Esempio 4.1.1.

5.2 La Gerarchia di Chomsky

Chomsky classificò le grammatiche in quattro tipi in base alla forma delle loro produzioni, ottenendo una gerarchia di classi di linguaggi contenute l'una nell'altra.

Definizione 5.2.1 (Grammatiche di Tipo 0 - Ricorsivamente Enumerabili): Una grammatica $G = (V, \Sigma, P, S)$ è di Tipo 0 (o senza restrizioni) se non vi sono restrizioni sulla forma delle produzioni, ovvero $P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$.

I linguaggi generati da grammatiche di Tipo 0 sono detti ricorsivamente enumerabili.

Definizione 5.2.2 (Grammatiche di Tipo 1 - Context-Sensitive): Una grammatica $G = (V, \Sigma, P, S)$ è di Tipo 1 (o context-sensitive) se ogni produzione in P è della forma:

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

dove $A \in V$, $\alpha, \beta \in (V \cup \Sigma)^*$, $\gamma \in (V \cup \Sigma)^+$.

Alternativamente, una grammatica è context-sensitive se ogni produzione $\alpha \rightarrow \beta$ soddisfa $|\alpha| \leq |\beta|$, con la possibile eccezione di $S \rightarrow \epsilon$ se S non appare nella parte destra di alcuna produzione.

La condizione $|\alpha| \leq |\beta|$ è detta condizione di non-contrazione, e garantisce che le derivazioni non riducano mai la lunghezza delle stringhe (eccetto possibilmente nel primo passo se $S \rightarrow \epsilon$).

Definizione 5.2.3 (Grammatiche di Tipo 2 - Context-Free): Una grammatica $G = (V, \Sigma, P, S)$ è di Tipo 2 (o context-free, libera dal contesto) se ogni produzione in P è della forma:

$$A \rightarrow \gamma$$

dove $A \in V$ e $\gamma \in (V \cup \Sigma)^*$.

Quindi in una grammatica context-free, la parte sinistra di ogni produzione è un singolo non terminale.

Definizione 5.2.4 (Grammatiche di Tipo 3 - Regolari): Una grammatica $G = (V, \Sigma, P, S)$ è di Tipo 3 (o regolare) se ogni produzione in P è di una delle seguenti forme:

- $A \rightarrow aB$ (regolare destra)
- $A \rightarrow a$ (regolare destra)
- $A \rightarrow \epsilon$ (solo per S , se S non appare in parte destra)

oppure

- $A \rightarrow Ba$ (regolare sinistra)
- $A \rightarrow a$ (regolare sinistra)
- $A \rightarrow \epsilon$ (solo per S , se S non appare in parte destra)

dove $A, B \in V$ e $a \in \Sigma$.

Una grammatica si dice regolare destra (o lineare destra) se tutte le produzioni sono di forma $A \rightarrow aB$ o $A \rightarrow a$. Analogamente per grammatiche regolari sinistre (o lineari sinistre).

Teorema 5.2.1 (Gerarchia di Chomsky): Siano $\mathcal{L}_3, \mathcal{L}_2, \mathcal{L}_1, \mathcal{L}_0$ rispettivamente le classi dei linguaggi generati da grammatiche di Tipo 3, 2, 1, 0. Allora:

$$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$$

Inoltre, tutte le inclusioni sono proprie.

La dimostrazione completa di questo teorema richiede la costruzione di linguaggi in ciascuna classe che non appartengono alla classe inferiore. Forniremo dimostrazioni parziali e esempi nel seguito.

Teorema 5.2.2 (Equivalenza tra grammatiche regolari e linguaggi regolari): Un linguaggio L è regolare (ovvero riconoscibile da un automa a stati finiti) se e solo se esiste una grammatica regolare G tale che $L = L(G)$.

Dimostrazione: Dimostriamo le due direzioni.

($\Leftarrow$) Sia $G = (V, \Sigma, P, S)$ una grammatica regolare destra. Costruiamo un NFA $N = (Q, \Sigma, \Delta, \{S\}, F)$ dove:

- $Q = V \cup \{q_{\text{acc}}\}$ dove q_{acc} è un nuovo stato accettante
- $F = \{q_{\text{acc}}\} \cup \{S\}$ se $S \rightarrow \varepsilon \in P$, altrimenti $F = \{q_{\text{acc}}\}$
- Le transizioni Δ sono definite da:
 - Se $A \rightarrow aB \in P$, allora $B \in \Delta(A, a)$
 - Se $A \rightarrow a \in P$, allora $q_{\text{acc}} \in \Delta(A, a)$

Dimostriamo per induzione su n che $A \Rightarrow^n w$ se e solo se $\delta^*(A, w)$ contiene q_{acc} o $w = \varepsilon$ e $A \in F$.

Base ($n=0$): Se $A \Rightarrow^0 w$, allora $w = A$, che non è una stringa terminale a meno che $w = \varepsilon$ e $A = S$ con $S \rightarrow \varepsilon \in P$.

Passo: Se $A \Rightarrow^{n+1} w$, allora la prima derivazione è $A \rightarrow aB$ (o $A \rightarrow a$) e $B \Rightarrow^n v$ dove $w = av$. Per ipotesi induttiva, $q_{\text{acc}} \in \delta^*(B, v)$. Poiché $B \in \Delta(A, a)$, abbiamo $q_{\text{acc}} \in \delta^*(A, av) = \delta^*(A, w)$.

($\Rightarrow$) Sia $A = (Q, \Sigma, \delta, q_0, F)$ un DFA che riconosce L . Costruiamo una grammatica regolare $G = (V, \Sigma, P, q_0)$ dove:

- $V = Q$
- Per ogni $q \in Q$ e $a \in \Sigma$ con $\delta(q, a) = p$, aggiungiamo la produzione $q \rightarrow ap$
- Per ogni $q \in F$, aggiungiamo la produzione $q \rightarrow \varepsilon$

Si dimostra per induzione che $q \Rightarrow^* w$ se e solo se $\delta^*(q, w) \in F$.

Base: $q \Rightarrow^0 \varepsilon$ se e solo se $q \rightarrow \varepsilon \in P$ se e solo se $q \in F$ se e solo se $\delta^*(q, \varepsilon) = q \in F$.

Passo: Se $q \Rightarrow^* w$ con $w = av$, allora la prima derivazione è $q \rightarrow ap$ per qualche $p = \delta(q, a)$, e $p \Rightarrow^* v$. Per ipotesi induttiva, $\delta^*(p, v) \in F$. Quindi $\delta^*(q, av) = \delta^*(\delta(q, a), v) = \delta^*(p, v) \in F$.

Quindi $L(G) = \{w : q_0 \Rightarrow^* w\} = \{w : \delta^*(q_0, w) \in F\} = L(A)$. ■

Esempio 5.2.1: La grammatica regolare con produzioni:

- $S \rightarrow 0S \mid 1S \mid 0A$
- $A \rightarrow 1B$
- $B \rightarrow \varepsilon$

genera il linguaggio di tutte le stringhe binarie contenenti la sottostringa 01. Questo corrisponde all'espressione regolare $(0 + 1)01(0 + 1)$.

5.3 Grammatiche Context-Free: Definizioni e Proprietà Fondamentali

Le grammatiche context-free (CFG) sono di particolare importanza sia nella teoria dei linguaggi formali che nelle applicazioni pratiche, in quanto catturano la struttura sintattica di molti linguaggi di programmazione e aspetti significativi delle lingue naturali.

Definizione 5.3.1 (Forma sentenziale): Sia $G = (V, \Sigma, P, S)$ una CFG. Una forma sentenziale è una stringa $\alpha \in (V \cup \Sigma)^*$ tale che $S \Rightarrow^* \alpha$.

Definizione 5.3.2 (Derivazione leftmost e rightmost): Una derivazione è leftmost (o più a sinistra) se in ogni passo si riscrive sempre il non terminale più a sinistra. Analogamente, una derivazione è rightmost (o più a destra) se si riscrive sempre il non terminale più a destra.

Scriviamo $\alpha \Rightarrow_{lm} \beta$ per una derivazione leftmost diretta, e $\alpha \Rightarrow_{lm} \beta$ per una derivazione leftmost in zero o più passi. Analogamente per $\Rightarrow_{rm}$ e $\Rightarrow_{rm}$.

Esempio 5.3.1: Consideriamo la grammatica G con produzioni:

- $S \rightarrow S + S \mid S * S \mid (S) \mid a$

Una derivazione leftmost di $a + a * a$ è:

$S \Rightarrow_{lm} S + S \Rightarrow_{lm} a + S \Rightarrow_{lm} a + S * S \Rightarrow_{lm} a + a * S \Rightarrow_{lm} a + a * a$

Una derivazione rightmost della stessa stringa è:

$S \Rightarrow_{rm} S + S \Rightarrow_{rm} S + S * S \Rightarrow_{rm} S + S * a \Rightarrow_{rm} S + a * a \Rightarrow_{rm} a + a * a$

Definizione 5.3.3 (Albero di derivazione): Sia $G = (V, \Sigma, P, S)$ una CFG. Un albero di derivazione (o parse tree) è un albero ordinato tale che:

- (i) Ogni nodo interno è etichettato con un non terminale di V
- (ii) Ogni foglia è etichettata con un simbolo di $\Sigma \cup \{\varepsilon\}$
- (iii) Se un nodo interno è etichettato con A e ha figli etichettati con $X_1, X_2, \dots, X_n$ (da sinistra a destra), allora $A \rightarrow X_1 X_2 \dots X_n \in P$
- (iv) La radice è etichettata con S
- (v) Se una foglia è etichettata con ε , allora è l'unico figlio del suo genitore

La frontiera (o yield) di un albero di derivazione è la stringa ottenuta concatenando le etichette delle foglie da sinistra a destra.

Lemma 5.3.1: Sia G una CFG. Allora $w \in L(G)$ se e solo se esiste un albero di derivazione per G con frontiera w .

Dimostrazione: ($\Rightarrow$) Se $S \Rightarrow^* w$, costruiamo un albero di derivazione per induzione sul numero di passi nella derivazione.

($\Leftarrow$) Se esiste un albero di derivazione con frontiera w , dimostriamo per induzione sull'altezza dell'albero che la radice (etichettata con S) deriva w . ■

Definizione 5.3.4 (Ambiguità): Una stringa $w \in L(G)$ è ambigua se ammette due o più alberi di derivazione distinti. Una grammatica G è ambigua se esiste almeno una stringa ambigua in $L(G)$. Un linguaggio L è intrinsecamente ambiguo se ogni grammatica che genera L è ambigua.

Esempio 5.3.2: La grammatica dell'Esempio 5.3.1 è ambigua. La stringa $a + a * a$ ha (almeno) due alberi di derivazione distinti corrispondenti alle strutture:

- $(a + a) * a$
- $a + (a * a)$

Questo riflette l'assenza di precedenza tra gli operatori $+$ e $*$.

Definizione 5.3.5 (Grammatica in Forma Normale di Chomsky): Una CFG $G = (V, \Sigma, P, S)$ è in Forma Normale di Chomsky (CNF) se ogni produzione è di una delle seguenti forme:

(i) $A \rightarrow BC$ dove $B, C \in V$ (ii) $A \rightarrow a$ dove $a \in \Sigma$ (iii) $S \rightarrow \varepsilon$ (solo se $\varepsilon \in L(G)$)

e inoltre S non appare nella parte destra di alcuna produzione se $S \rightarrow \varepsilon \in P$.

Teorema 5.3.2: Ogni linguaggio context-free può essere generato da una grammatica in Forma Normale di Chomsky.

Dimostrazione: Sia $G = (V, \Sigma, P, S)$ una CFG. Costruiamo una grammatica equivalente G' in CNF attraverso i seguenti passi:

Passo 1 (Eliminazione di ε -produzioni): Una produzione $A \rightarrow \varepsilon$ è detta ε -produzione. Un non terminale A è nullable se $A \Rightarrow^* \varepsilon$.

Algoritmo per eliminare le ε -produzioni:

1. Calcola l'insieme N di tutti i non terminali nullable
2. Per ogni produzione $A \rightarrow X_1 X_2 \dots X_k$, aggiungi tutte le produzioni $A \rightarrow \gamma$ dove γ si ottiene da $X_1 X_2 \dots X_k$ omettendo un sottoinsieme qualsiasi degli $X_i \in N$ (escludendo il caso in cui si omettono tutti)
3. Rimuovi tutte le ε -produzioni eccetto eventualmente $S \rightarrow \varepsilon$ se $\varepsilon \in L(G)$

Passo 2 (Eliminazione delle produzioni unitarie): Una produzione $A \rightarrow B$ dove $B \in V$ è detta unitaria.

Algoritmo per eliminare le produzioni unitarie:

1. Per ogni coppia (A, B) tale che $A \Rightarrow^* B$ usando solo produzioni unitarie, aggiungi tutte le produzioni $A \rightarrow \gamma$ dove $B \rightarrow \gamma$ è una produzione non unitaria
2. Rimuovi tutte le produzioni unitarie

Passo 3 (Conversione in CNF): Per ogni produzione $A \rightarrow X_1 X_2 \dots X_k$ con $k \geq 2$:

1. Se $X_i = a \in \Sigma$, introduci un nuovo non terminale C_a con produzione $C_a \rightarrow a$, e sostituisci a con C_a nella produzione originale
2. Se $k > 2$, sostituisci $A \rightarrow X_1 X_2 \dots X_k$ con:
 - $A \rightarrow X_1 A_1$
 - $A_1 \rightarrow X_2 A_2$
 - ...
 - $A_{k-2} \rightarrow X_{k-1} X_k$ dove $A_1, A_2, \dots, A_{k-2}$ sono nuovi non terminali

Si può verificare che ogni passo preserva il linguaggio generato e produce una grammatica più vicina alla CNF. ■

Esempio 5.3.3: Convertiamo la grammatica:

- $S \rightarrow ASA \mid aB$
- $A \rightarrow B \mid S$
- $B \rightarrow b \mid \varepsilon$

Passo 1: B è nullable, quindi anche A è nullable, e quindi anche S è nullable. Nuove produzioni derivate:

- $S \rightarrow ASA \mid AS \mid SA \mid S \mid aB \mid a$
- $A \rightarrow B \mid S$
- $B \rightarrow b$

Passo 2: Abbiamo le coppie unitarie (A,B) , (A,S) , (S,A) , (S,B) , e le loro composizioni. Nuove produzioni:

- $S \rightarrow ASA \mid AS \mid SA \mid aB \mid a \mid b$
- $A \rightarrow b \mid ASA \mid AS \mid SA \mid aB \mid a$
- $B \rightarrow b$

Passo 3: Introduciamo $C_a \rightarrow a$ e convertiamo:

- $S \rightarrow AA_1 \mid AS \mid SA \mid C_a B \mid a \mid b$
- $A_1 \rightarrow SA$
- $A \rightarrow b \mid AA_2 \mid AS \mid SA \mid C_a B \mid a$
- $A_2 \rightarrow SA$
- $B \rightarrow b$
- $C_a \rightarrow a$

Questa grammatica è ora in CNF.

Definizione 5.3.6 (Forma Normale di Greibach): Una CFG è in Forma Normale di Greibach (GNF) se ogni produzione è della forma:

$A \rightarrow a\alpha$

dove $a \in \Sigma$ e $\alpha \in V^*$.

Teorema 5.3.3: Ogni linguaggio context-free senza ε può essere generato da una grammatica in Forma Normale di Greibach.

La dimostrazione della conversione in GNF è tecnica e richiede la risoluzione di sistemi di equazioni grammaticali. Omettiamo i dettagli.

5.4 Automi a Pila e Teorema di Equivalenza

Gli automi a pila (pushdown automata, PDA) sono la controparte riconoscitiva delle grammatiche context-free, così come gli automi a stati finiti sono la controparte delle grammatiche regolari.

Definizione 5.4.1 (Automa a pila non deterministico): Un automa a pila non deterministico (NPDA) è una settupla $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ dove:

(i) Q è un insieme finito di stati (ii) Σ è l'alfabeto di input (iii) Γ è l'alfabeto della pila (iv) $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}_{\text{fin}}(Q \times \Gamma^*)$ è la funzione di transizione, dove $\mathcal{P}_{\text{fin}}$ denota l'insieme delle parti finite (v) $q_0 \in Q$ è lo stato iniziale (vi) $Z_0 \in \Gamma$ è il simbolo iniziale della pila (vii) $F \subseteq Q$ è l'insieme degli stati finali

Definizione 5.4.2 (Configurazione): Una configurazione (o descrizione istantanea) di un PDA è una tripla (q, w, γ) dove:

- $q \in Q$ è lo stato corrente
- $w \in \Sigma^*$ è la parte rimanente dell'input
- $\gamma \in \Gamma^*$ è il contenuto corrente della pila (con il simbolo in cima a sinistra)

Definizione 5.4.3 (Relazione di transizione): Definiamo la relazione $\vdash_M$ (o semplicemente $\vdash$) tra configurazioni come segue:

$(q, aw, Z\beta) \vdash (p, w, \alpha\beta)$

se e solo se $(p, \alpha) \in \delta(q, a, Z)$, dove $a \in \Sigma \cup \{\varepsilon\}$.

Intuitivamente, il PDA nello stato q , leggendo il simbolo a (o ε per una ε -transizione) e con Z in cima alla pila, può passare allo stato p , consumare a (se $a \neq \varepsilon$), rimpiazzare Z con α .

Come al solito, $\vdash^*$ denota la chiusura riflessiva e transitiva di $\vdash$.

Definizione 5.4.4 (Linguaggio accettato): Esistono due modalità di accettazione per i PDA:

(i) Accettazione per stato finale: $L(M) = \{w \in \Sigma^* : (q_0, w, Z_0) \vdash^* (q, \varepsilon, \gamma) \text{ per qualche } q \in F, \gamma \in \Gamma^*\}$

(ii) Accettazione per pila vuota: $N(M) = \{w \in \Sigma^* : (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon) \text{ per qualche } q \in Q\}$

Teorema 5.4.1: Le due modalità di accettazione sono equivalenti, nel senso che per ogni PDA M_1 che accetta per stato finale esiste un PDA M_2 che accetta per pila vuota tale che $L(M_1) = N(M_2)$, e viceversa.

Dimostrazione:

Direzione 1: Dato $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_0, Z_0, F_1)$, costruiamo $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q'_0, Z'_0, \emptyset)$ dove:

- $Q_2 = Q_1 \cup \{q'_0, q_empty\}$
- $\Gamma_2 = \Gamma_1 \cup \{Z'_0\}$
- δ_2 include:
 - $\delta_2(q'_0, \epsilon, Z'_0) = \{(q_0, Z_0 Z'_0)\}$ (inizializza M_1 preservando Z'_0 sul fondo)
 - $\delta_2(q, a, Z) = \delta_1(q, a, Z)$ per $q \in Q_1$ (simula M_1)
 - $\delta_2(q, \epsilon, Z)$ contiene (q_empty, ϵ) per ogni $q \in F_1$ e $Z \in \Gamma_2$ (svuota la pila se in stato finale)
 - $\delta_2(q_empty, \epsilon, Z) = \{(q_empty, \epsilon)\}$ per ogni $Z \in \Gamma_2$ (continua a svuotare)

Si verifica che $w \in L(M_1)$ se e solo se $w \in N(M_2)$.

Direzione 2: Dato M_2 che accetta per pila vuota, la costruzione di M_1 è simile ma inversa. ■

Teorema 5.4.2 (Equivalenza CFG-PDA): Un linguaggio L è context-free se e solo se esiste un PDA M tale che $L = L(M)$.

Dimostrazione: Dimostriamo entrambe le direzioni.

($\Leftarrow$) Dato un PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, costruiamo una CFG $G = (V, \Sigma, P, S)$ che simula M . L'idea chiave è che i non terminali di G codificano coppie di stati di M e il contenuto della pila tra questi stati.

Per semplicità, assumiamo che M accetti per pila vuota e che ogni transizione o spinge esattamente un simbolo sulla pila o ne preleva esattamente uno.

I non terminali di G hanno la forma $[q, A, p]$ dove $q, p \in Q$ e $A \in \Gamma$, con l'interpretazione che $[q, A, p]$ deriva tutte le stringhe w tali che M può andare dallo stato q con A in cima alla pila allo stato p con la pila svuotata di A , leggendo w .

Le produzioni sono:

- $S \rightarrow [q_0, Z_0, q]$ per ogni $q \in Q$
- Se $(p, B) \in \delta(q, a, A)$ (M legge a , preleva A , spinge B , va in p), allora aggiungiamo: $[q, A, s] \rightarrow a[p, B, s]$ per ogni $s \in Q$
- Se $(p, \epsilon) \in \delta(q, a, A)$ (M legge a , preleva A , va in p), allora aggiungiamo: $[q, A, p] \rightarrow a$

Si dimostra per induzione che $[q, A, p] \Rightarrow^* w$ se e solo se $(q, w, A) \vdash^* (p, \epsilon, \epsilon)$.

($\Rightarrow$) Data una CFG $G = (V, \Sigma, P, S)$ in CNF, costruiamo un PDA $M = (\{q\}, \Sigma, V \cup \Sigma, \delta, q, S, \emptyset)$ con un solo stato che accetta per pila vuota:

- Se $A \rightarrow BC \in P$, allora $\delta(q, \varepsilon, A)$ contiene (q, CB) (nota: ordine invertito)
- Se $A \rightarrow a \in P$, allora $\delta(q, a, A)$ contiene (q, ε)

Il PDA simula una derivazione leftmost di G : mantiene la parte rimanente della forma sentenziale sulla pila e la confronta con l'input.

Si dimostra che $S \Rightarrow_{lm} w$ se e solo se $(q, w, S) \vdash (q, \varepsilon, \varepsilon)$. ■

Corollario 5.4.3: I linguaggi context-free sono chiusi rispetto alle operazioni di unione, concatenazione e chiusura di Kleene.

Dimostrazione: Date CFG G_1, G_2 per L_1, L_2 (con non terminali disgiunti), costruiamo:

- Unione: Nuova grammatica con simbolo iniziale S e produzioni $S \rightarrow S_1 \mid S_2$
- Concatenazione: Nuovo simbolo iniziale S con produzione $S \rightarrow S_1 S_2$
- Chiusura: Nuovo simbolo iniziale S con produzioni $S \rightarrow S_1 S \mid \varepsilon$

dove S_1, S_2 sono i simboli iniziali di G_1, G_2 . ■

Teorema 5.4.4 (Non chiusura sotto intersezione): I linguaggi context-free non sono chiusi rispetto all'intersezione.

Dimostrazione: Consideriamo:

- $L_1 = \{a^n b^n c^m : n, m \geq 0\}$ (context-free)
- $L_2 = \{a^m b^n c^n : n, m \geq 0\}$ (context-free)

Entrambi sono context-free (esercizio: fornire le grammatiche). Tuttavia:

$$L_1 \cap L_2 = \{a^n b^n c^n : n \geq 0\}$$

Questo linguaggio non è context-free, come dimostreremo con il lemma di pumping per CFG (Teorema 5.5.1). ■

Teorema 5.4.5 (Chiusura sotto intersezione con linguaggi regolari): Se L è context-free e R è regolare, allora $L \cap R$ è context-free.

Dimostrazione: Siano $M = (Q_M, \Sigma, \Gamma, \delta_M, q_M, Z_0, F_M)$ un PDA per L e $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ un DFA per R . Costruiamo un PDA $M' = (Q', \Sigma, \Gamma, \delta', (q_M, q_A), Z_0, F')$ dove:

- $Q' = Q_M \times Q_A$
- $F' = F_M \times F_A$
- $\delta'((p, q), a, Z) = \{(p', q') : (p', \gamma) \in \delta_M(p, a, Z) \text{ e } q' = \delta_A(q, a)\}$

Il PDA M' simula M e A in parallelo. Una stringa è accettata se e solo se è accettata da entrambi, quindi $L(M') = L(M) \cap L(A) = L \cap R$. ■

Trattato sulla Teoria Formale delle Lingue Naturali

Parte IV: Proprietà Avanzate dei Linguaggi Context-Free

Capitolo 6: Lemma di Pumping e Proprietà Decisionali

6.1 Lemma di Pumping per Linguaggi Context-Free

Analogamente ai linguaggi regolari, i linguaggi context-free ammettono un lemma di pumping che fornisce una condizione necessaria (ma non sufficiente) per la context-freeness.

Teorema 6.1.1 (Lemma di Pumping per linguaggi context-free): Sia L un linguaggio context-free. Allora esiste una costante p (la lunghezza di pumping) tale che ogni stringa $z \in L$ con $|z| \geq p$ può essere scritta come $z = uvwxy$ dove:

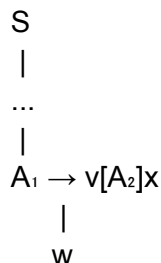
(i) $|vwx| \leq p$ (ii) $|vx| \geq 1$ (iii) Per ogni $i \geq 0$, $uv^iwx^iy \in L$

Dimostrazione: Sia $G = (V, \Sigma, P, S)$ una CFG in Forma Normale di Chomsky che genera L . Sia $k = |V|$ il numero di non terminali. Scegliamo $p = 2^k$.

Sia $z \in L$ con $|z| \geq p$. Poiché G è in CNF, ogni albero di derivazione per z è un albero binario. Se tutte le foglie sono a distanza al massimo k dalla radice, allora l'albero ha al massimo 2^k foglie, quindi $|z| \leq 2^k = p$, contraddicendo $|z| \geq p$. Quindi esiste almeno un cammino dalla radice a una foglia di lunghezza maggiore di k .

Consideriamo un tale cammino di lunghezza massima, e sia π il subcammino costituito dagli ultimi $k+1$ nodi (escludendo la foglia). Poiché ci sono k non terminali e π contiene $k+1$ nodi interni, per il principio del cassetto esiste un non terminale A che appare (almeno) due volte in π .

Siano A_1 e A_2 le due occorrenze di A in π , con A_2 più vicina alla foglia. Decomponiamo l'albero di derivazione come segue:



dove $A_1 \Rightarrow^* vA_2x \Rightarrow^* vwx$. Più precisamente:

- $S \Rightarrow^* uA_1y$ dove u e y sono le parti generate al di fuori del sottoalbero radicato in A_1
- $A_1 \Rightarrow^* vA_2x$
- $A_2 \Rightarrow^* w$

Quindi $z = uvwxy$.

Verifichiamo le condizioni:

(i) Poiché A_2 si trova a profondità al massimo $k+1$ dal nodo A_1 , e l'albero è in CNF (quindi binario), il sottoalbero radicato in A_1 ha al massimo 2^{k+1} foglie. Quindi $|vwx| \leq 2^{k+1} = 2p$.

In realtà, possiamo affinare la dimostrazione scegliendo A_2 come l'occorrenza di A più vicina a una foglia negli ultimi $k+1$ nodi. Allora il sottoalbero radicato in A_1 ha altezza al massimo $k+1$, quindi al massimo 2^{k+1} foglie, ma se consideriamo che A_2 è negli ultimi k nodi sotto A_1 , otteniamo $|vwx| \leq 2^k = p$. (La dimostrazione standard richiede maggior cura nella scelta del cammino.)

(ii) Se $v = x = \varepsilon$, allora i sottoalberi radicati in A_1 e A_2 sono identici, ma ciò viola la condizione che abbiamo scelto un cammino di lunghezza massima. Quindi almeno uno tra v e x è non vuoto, ovvero $|vx| \geq 1$.

(iii) Poiché $A_1 \Rightarrow^* vA_2x$ e $A_2 \Rightarrow^* w$, possiamo "iterare" la derivazione $A_1 \Rightarrow^* vA_2x$:

$$A_1 \Rightarrow^* vA_2x \Rightarrow^* vvA_2xx \Rightarrow^* vv\dots vA_2xx\dots x \text{ (i volte)} \Rightarrow^* v^i wx^i$$

Per $i = 0$, omettiamo completamente il sottoalbero vA_2x e usiamo direttamente $A_1 \Rightarrow^* A_2 \Rightarrow^* w$, ottenendo $A_1 \Rightarrow^* w$.

Quindi $S \Rightarrow^* uA_1y \Rightarrow^* uv^i A_2^i x^i y \Rightarrow^* uv^i wx^i y$ per ogni $i \geq 0$, e $uv^i wx^i y \in L(G) = L$. ■

Esempio 6.1.1: Dimostriamo che $L = \{a^n b^n c^n : n \geq 0\}$ non è context-free.

Dimostrazione: Supponiamo per contraddizione che L sia context-free con costante di pumping p . Consideriamo la stringa $z = a^p b^p c^p \in L$. Per il lemma di pumping, $z = uvwxy$ dove $|vwx| \leq p$, $|vx| \geq 1$, e $uv^i wx^i y \in L$ per ogni $i \geq 0$.

Poiché $|vwx| \leq p$, la sottostringa vwx non può contenere tutte e tre le lettere a , b , c . Distinguiamo i casi:

Caso 1: vwx contiene solo a e b (o solo b e c , o solo a e c - gli altri casi sono analoghi).

Allora v e x contengono solo a e b (almeno uno dei due è non vuoto). Considerando $i = 2$, la stringa uv^2wx^2y avrà più a e b rispetto a z , ma lo stesso numero di c . Quindi non può appartenere a L .

Caso 2: vwx contiene solo un tipo di lettera.

Allora v e x contengono solo lettere dello stesso tipo. Considerando $i = 2$, avremo un incremento di quel tipo di lettera, rompendo il bilanciamento. Quindi $uv^2wx^2y \notin L$.

In entrambi i casi otteniamo una contraddizione. Quindi L non è context-free. ■

Esempio 6.1.2: Il linguaggio $L = \{ww : w \in \{a, b\}^*\}$ non è context-free.

Dimostrazione: Sia p la costante di pumping. Consideriamo $z = a^p b^p a^p b^p \in L$. Scriviamo $z = uvwxy$ con le condizioni del lemma.

Poiché $|vwx| \leq p$, la sottostringa vwx si trova interamente in una delle quattro regioni a^p , b^p , a^p , b^p della stringa z .

Caso 1: vwx è nella prima metà ($a^p b^p$).

Allora $v^i wx^i$ è ancora nella prima metà per ogni i . Ma affinché $uv^i wx^i y \in L$, deve essere della forma ww per qualche w . Questo significa che la seconda metà deve cambiare esattamente come la prima, ma la seconda metà è fissa (uguale a $a^p b^p$). Contraddizione a meno che $v = x = \epsilon$, ma ciò viola $|vx| \geq 1$.

Caso 2: vwx è nella seconda metà.

Simmetrico al Caso 1.

Caso 3: vwx attraversa il confine tra prima e seconda metà.

Allora vwx è una sottostringa di $b^p a^p$. Pommando, otterremo uv^2wx^2y che ha più b nella parte centrale e più a nella parte centrale, rompendo la simmetria necessaria per essere in L . ■

6.2 Proprietà di Chiusura dei Linguaggi Context-Free

Abbiamo già visto che i linguaggi context-free sono chiusi sotto unione, concatenazione e stella, ma non sotto intersezione né complemento.

Teorema 6.2.1 (Chiusura sotto sostituzione context-free): Sia $L \subseteq \Sigma^*$ un linguaggio context-free e sia $s : \Sigma \rightarrow \mathcal{A}(\Gamma^*)$ una sostituzione tale che $s(a)$ è context-free per ogni $a \in \Sigma$. Allora $s(L)$ è context-free.

Dimostrazione: Sia $G = (V, \Sigma, P, S)$ una CFG per L , e per ogni $a \in \Sigma$ sia $G_a = (V_a, \Gamma, P_a, S_a)$ una CFG per $s(a)$ (assumiamo che tutti i V_a e V siano disgiunti).

Costruiamo una nuova grammatica $G' = (V', \Gamma, P', S)$ dove:

- $V' = V \cup \bigcup \{V_a : a \in \Sigma\}$
- $P' = (P \text{ con ogni terminale } a \text{ sostituito da } S_a) \cup \bigcup \{P_a : a \in \Sigma\}$

Si verifica che $L(G') = s(L(G)) = s(L)$. ■

Teorema 6.2.2 (Chiusura sotto omomorfismo): Se L è context-free e h è un omomorfismo, allora $h(L)$ è context-free.

Dimostrazione: Un omomorfismo è un caso particolare di sostituzione dove $s(a) = \{h(a)\}$ per ogni a . Quindi il risultato segue dal teorema precedente. ■

Teorema 6.2.3 (Chiusura sotto omomorfismo inverso): Se L è context-free e h è un omomorfismo, allora $h^{-1}(L)$ è context-free.

Dimostrazione: Sia $M = (Q, \Gamma, \Delta, \delta, q_0, Z_0, F)$ un PDA per L su alfabeto Γ , e sia $h : \Sigma^* \rightarrow \Gamma^*$ un omomorfismo.

Costruiamo un PDA $M' = (Q, \Sigma, \Delta, \delta', q_0, Z_0, F)$ dove δ' simula M processando $h(a)$ ogni volta che legge il simbolo a :

Per ogni $a \in \Sigma$, se $h(a) = b_1 b_2 \dots b_k$, definiamo $\delta'(q, a, Z)$ introducendo stati intermedi che processano b_1 , poi b_2 , ..., fino a b_k , mantenendo la pila come fa δ .

Formalmente, per ogni transizione $(p, \gamma) \in \delta(q, b, Z)$, introduciamo stati $q_{b1}, q_{b2}, \dots, q_{b(k-1)}$ e transizioni:

- $\delta'(q, a, Z)$ contiene (q_{b1}, Z) se $k \geq 1$
- $\delta'(q_{bi}, \epsilon, Z)$ contiene $(q_{b(i+1)}, Z)$ per $i = 1, \dots, k-2$
- $\delta'(q_{b(k-1)}, \epsilon, Z)$ contiene (p, γ)

Se $h(a) = \epsilon$, allora $\delta'(q, a, Z) = \{(q, Z)\}$ (nessun cambiamento).

Si dimostra che M' accetta w se e solo se M accetta $h(w)$, quindi $L(M') = h^{-1}(L(M)) = h^{-1}(L)$. ■

Teorema 6.2.4 (Chiusura sotto quoziente con linguaggi regolari): Se L è context-free e R è regolare, allora L / R è context-free.

Dimostrazione: Combiniamo il PDA per L con un DFA per R in modo analogo alla dimostrazione del Teorema 5.4.5, ma modificando gli stati finali per caratterizzare il quoziente. ■

Teorema 6.2.5 (Non chiusura sotto complemento): I linguaggi context-free non sono chiusi sotto complemento.

Dimostrazione: Se i CFG fossero chiusi sotto complemento, allora sarebbero chiusi anche sotto intersezione, poiché $L_1 \cap L_2 = \neg(\neg L_1 \cup \neg L_2)$. Ma sappiamo dal Teorema 5.4.4 che non sono chiusi sotto intersezione. ■

6.3 Automi a Pila Deterministici

Gli automi a pila deterministici (DPDA) sono una sottoclasse importante dei PDA che corrispondono ai linguaggi context-free deterministici, rilevanti per l'analisi sintattica efficiente.

Definizione 6.3.1 (Automa a pila deterministico): Un PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ è deterministico (DPDA) se:

- (i) Per ogni $q \in Q$, $a \in \Sigma \cup \{\epsilon\}$, $Z \in \Gamma$, si ha $|\delta(q, a, Z)| \leq 1$
- (ii) Se $\delta(q, \epsilon, Z) \neq \emptyset$, allora $\delta(q, a, Z) = \emptyset$ per ogni $a \in \Sigma$

La condizione (i) garantisce che in ogni configurazione esista al massimo una transizione applicabile. La condizione (ii) garantisce che non vi sia scelta tra fare una ε -transizione o leggere un simbolo di input.

Definizione 6.3.2 (Linguaggio context-free deterministico): Un linguaggio L è context-free deterministico (DCFL) se esiste un DPDA M tale che $L = L(M)$ (con accettazione per stato finale).

Teorema 6.3.1: La classe DCFL è propriamente contenuta nella classe dei linguaggi context-free.

Dimostrazione: Chiaramente $DCFL \subseteq CFL$ poiché ogni DPDA è un PDA. Per mostrare che l'inclusione è propria, consideriamo:

$$L = \{a^n b^n : n \geq 0\} \cup \{a^n b^{2n} : n \geq 0\}$$

Questo linguaggio è context-free (unione di due CFG), ma non è DCFL. La ragione intuitiva è che un DPDA, leggendo a^n , non può decidere deterministicamente se aspettarsi b^n o b^{2n} senza guardare avanti l'intero blocco di b .

Una dimostrazione formale richiede il teorema che afferma che i DCFL sono chiusi sotto complemento (Teorema 6.3.3), mentre L e il suo complemento possono essere mostrati incompatibili con questa proprietà. ■

Teorema 6.3.2 (Proprietà di prefisso): Se L è accettato da un DPDA per stato finale, allora L ha la proprietà del prefisso: nessuna stringa in L è prefisso proprio di un'altra stringa in L .

Dimostrazione: Supponiamo per contraddizione che esistano $w, wv \in L$ con $v \neq \varepsilon$. Poiché M è deterministico, esiste un'unica computazione su w :

$$(q_0, w, Z_0) \vdash^* (q_f, \varepsilon, \gamma)$$

con $q_f \in F$. Continuando la computazione su wv :

$$(q_0, wv, Z_0) \vdash^* (q_f, v, \gamma) \vdash^* (q'_f, \varepsilon, \gamma')$$

con $q'_f \in F$. Ma dopo aver letto w , M è nello stato q_f e ha già accettato. La lettura di v cambia lo stato da q_f a q'_f . Se $q_f = q'_f$ e $\gamma = \gamma'$, allora M è in un ciclo che non consuma input né modifica la pila, il che è impossibile se $v \neq \varepsilon$. Se $q_f \neq q'_f$ o $\gamma \neq \gamma'$, allora M ha cambiato configurazione dopo aver già accettato, il che contraddice il determinismo. ■

Corollario 6.3.2.1: Per ovviare alla proprietà di prefisso, spesso si considerano DPDA che accettano per pila vuota, oppure si aggiunge un marcatore di fine stringa $\$$.

Teorema 6.3.3 (Chiusura dei DCFL sotto complemento): La classe dei linguaggi context-free deterministici è chiusa rispetto al complemento.

Dimostrazione (Schema): Dato un DPDA M che accetta per pila vuota, costruiamo un DPDA M' per il complemento come segue:

1. Modifichiamo M per accettare sempre (processando tutto l'input e raggiungendo una configurazione unica di accettazione o rigetto)
2. Aggiungiamo transizioni per consumare tutto l'input rimanente se M si blocca
3. Scambiamo le configurazioni di accettazione con quelle di rigetto

La dimostrazione completa richiede cura nel trattare i casi in cui M può entrare in loop infinito o non consumare tutto l'input. Si introduce la nozione di DPDA "completo" che legge sempre tutto l'input. ■

Teorema 6.3.4: I DCFL non sono chiusi sotto unione, intersezione, concatenazione o chiusura di Kleene.

Dimostrazione: Controesempi:

- Per unione: $L_1 = \{a^n b^n c^m : n, m \geq 0\}$ e $L_2 = \{a^m b^n c^n : n, m \geq 0\}$ sono DCFL, ma $L_1 \cup L_2$ non è DCFL (può essere mostrato usando le proprietà di chiusura sotto complemento).
- Per intersezione: Segue dal fatto che i DCFL sono chiusi sotto complemento ma non sotto unione (usando le leggi di De Morgan).
- Per concatenazione e stella: Controesempi simili. ■

6.4 Problemi Decisionali per Linguaggi Context-Free

Definizione 6.4.1 (Problema della membership): Dato un linguaggio L e una stringa w , determinare se $w \in L$.

Teorema 6.4.1 (Decidibilità del problema della membership per CFL): Esiste un algoritmo che, data una CFG G e una stringa w , decide se $w \in L(G)$.

Dimostrazione: Presentiamo due algoritmi classici.

Algoritmo 1 (Algoritmo di Cocke-Younger-Kasami, CYK):

Assumiamo che $G = (V, \Sigma, P, S)$ sia in Forma Normale di Chomsky. Sia $w = a_1 a_2 \dots a_n$ con $n \geq 1$ (il caso $w = \epsilon$ si tratta separatamente).

Costruiamo una tabella triangolare T di dimensione $n \times n$ dove $T[i, j]$ contiene l'insieme dei non terminali A tali che $A \Rightarrow^* a_i a_{i+1} \dots a_{i+j-1}$ (la sottostringa di w che inizia alla posizione i e ha lunghezza j).

Inizializzazione ($j = 1$): Per $i = 1, 2, \dots, n$: $T[i, 1] = \{A \in V : A \rightarrow a_i \in P\}$

Riempimento ($j = 2, 3, \dots, n$): Per $i = 1, 2, \dots, n - j + 1$: $T[i, j] = \{A \in V : \text{esiste } A \rightarrow BC \in P \text{ e } 1 \leq k < j \text{ tale che } B \in T[i, k] \text{ e } C \in T[i+k, j-k]\}$

Accettazione: $w \in L(G)$ se e solo se $S \in T[1, n]$

Analisi di complessità: L'algoritmo ha tre cicli annidati (j, i, k), ciascuno che itera al massimo n volte. Per ogni tripla (i, j, k) , verifichiamo tutte le produzioni (al massimo $O(|P|)$). Quindi la complessità temporale è $O(n^3|P|)$, ovvero $O(n^3)$ per grammatica fissa.

Algoritmo 2 (Algoritmo di Earley):

L'algoritmo di Earley è più generale e funziona per grammatiche arbitrarie (non necessariamente in CNF), con complessità $O(n^3)$ nel caso peggiore, ma $O(n^2)$ per grammatiche non ambigue e $O(n)$ per grammatiche LR(k).

L'algoritmo mantiene insiemi di "item" della forma $[A \rightarrow \alpha \cdot \beta, i]$ dove $A \rightarrow \alpha\beta$ è una produzione, il punto $\cdot$ indica quanta parte della produzione è stata riconosciuta, e i è la posizione di partenza in w .

Omettiamo i dettagli tecnici, ma l'idea è di procedere da sinistra a destra su w , mantenendo un insieme S_j per ogni posizione j , contenente tutti gli item validi fino a quella posizione. ■

Definizione 6.4.2 (Problema del vuoto): Data una CFG G , determinare se $L(G) = \emptyset$.

Teorema 6.4.2 (Decidibilità del problema del vuoto): Esiste un algoritmo che decide se $L(G) = \emptyset$ per una CFG G .

Dimostrazione: Calcoliamo l'insieme dei non terminali "produttivi", ovvero quelli che derivano almeno una stringa terminale.

Algoritmo:

1. Inizializza $P_0 = \{A \in V : \text{esiste } A \rightarrow w \in P \text{ con } w \in \Sigma^*\}$
2. Per $i = 0, 1, 2, \dots$, finché $P_i \neq P_{i+1}$: $P_{i+1} = P_i \cup \{A \in V : \text{esiste } A \rightarrow \alpha \in P \text{ con } \alpha \in (P_i \cup \Sigma)^*\}$
3. $L(G) = \emptyset$ se e solo se $S \notin P_n$ (dove P_n è il punto fisso)

L'algoritmo termina in al massimo $|V|$ iterazioni poiché P_i cresce strettamente o si stabilizza. ■

Definizione 6.4.3 (Problema della finitezza): Data una CFG G , determinare se $L(G)$ è finito o infinito.

Teorema 6.4.3 (Decidibilità del problema della finitezza): Esiste un algoritmo che decide se $L(G)$ è infinito.

Dimostrazione: Convertiamo G in CNF e costruiamo un grafo di dipendenza tra non terminali: c'è un arco $A \rightarrow B$ se esiste una produzione $A \rightarrow BC$ o $A \rightarrow CB$ in G .

$L(G)$ è infinito se e solo se:

1. $L(G) \neq \emptyset$, e
2. Esiste un non terminale A raggiungibile da S e da cui si può raggiungere un terminale, tale che A è in un ciclo nel grafo di dipendenza.

La condizione 2 corrisponde all'esistenza di una derivazione $S \Rightarrow^* uAv \Rightarrow^* uA'Av \Rightarrow^* uw'v$ per qualche $u, v, w' \in \Sigma^*$, A' derivante da A , che permette di "pompare" la derivazione indefinitamente.

L'algoritmo verifica queste condizioni in tempo polinomiale usando ricerca in profondità sul grafo. ■

Definizione 6.4.4 (Problema dell'equivalenza): Date due CFG G_1 e G_2 , determinare se $L(G_1) = L(G_2)$.

Teorema 6.4.4 (Indecidibilità del problema dell'equivalenza): Non esiste un algoritmo che decide se $L(G_1) = L(G_2)$ per CFG arbitrarie G_1 e G_2 .

Dimostrazione (Schema): La dimostrazione si basa su una riduzione dal Post Correspondence Problem (PCP), che è noto essere indecidibile. Dato un'istanza del PCP, si costruiscono due CFG G_1 e G_2 tali che $L(G_1) = L(G_2)$ se e solo se il PCP ha soluzione.

Il PCP consiste di due liste di stringhe $(u_1, \dots, u_k)$ e $(v_1, \dots, v_k)$, e chiede se esiste una sequenza di indici $i_1, i_2, \dots, i_m$ tale che:

$$u_{\{i_1\}} u_{\{i_2\}} \dots u_{\{i_m\}} = v_{\{i_1\}} v_{\{i_2\}} \dots v_{\{i_m\}}$$

Si possono costruire CFG che generano tutte le possibili concatenazioni $u_{\{i_1\}} u_{\{i_2\}} \dots u_{\{i_m\}}$ e $v_{\{i_1\}} v_{\{i_2\}} \dots v_{\{i_m\}}$, codificando gli indici. Se i due linguaggi sono uguali, allora esiste una corrispondenza che soddisfa il PCP. ■

Corollario 6.4.4.1: Anche il problema dell'inclusione (decidere se $L(G_1) \subseteq L(G_2)$) è indecidibile per CFG.

Teorema 6.4.5: Il problema della membership per i DCFL è decidibile in tempo lineare $O(n)$.

Dimostrazione: Un DPDA ha al massimo una computazione per ogni input, quindi possiamo semplicemente simulare il DPDA per al più n passi (dove $n = |w|$). Se il DPDA accetta, allora $w \in L$; altrimenti $w \notin L$. La simulazione richiede tempo $O(n)$. ■

Teorema 6.4.6: Il problema dell'ambiguità (decidere se una CFG è ambigua) è indecidibile.

Dimostrazione (Schema): Anche questo si riduce al PCP. Data un'istanza del PCP, si costruisce una CFG tale che è ambigua se e solo se il PCP ha soluzione. L'idea è che le due diverse derivazioni per la stessa stringa corrispondano a due diverse soluzioni del PCP (o alla stessa soluzione trovata in due modi diversi). ■

6.5 Pumping Lemma per Linguaggi Lineari

Definizione 6.5.1 (Grammatica lineare): Una CFG G è lineare se ogni produzione ha la forma: $A \rightarrow uBv$ o $A \rightarrow w$ dove $u, v, w \in \Sigma^*$ e B è un singolo non terminale.

Teorema 6.5.1 (Pumping Lemma per linguaggi lineari): Se L è un linguaggio lineare, esiste p tale che ogni $z \in L$ con $|z| \geq p$ può essere scritta come $z = uvwx$ dove:

1. $|vwx| \leq p$
2. $|vx| \geq 1$
3. uv e wx hanno al più una occorrenza di non terminali ciascuna
4. $\forall i \geq 0: uv^iwx^iy \in L$

Dimostrazione: Simile al pumping lemma per CFL, ma sfruttando la struttura lineare delle derivazioni. La chiave è che in una grammatica lineare, ogni forma sentenziale ha al più un non terminale, quindi la "forma" del pumping è più restrittiva. ■

6.6 Proprietà Decidibili Aggiuntive per CFG

Teorema 6.6.1 (Decidibilità della regolarità): È decidibile se un linguaggio context-free è regolare.

Algoritmo:

```
is_regular(G):
  Convert G to CNF
  Construct dependency graph D where:
    A → B if ∃ production A → αBβ with α ≠ ε or β ≠ ε

  return TRUE iff no cycles exist in D
```

Teorema di correttezza: $L(G)$ è regolare $\Leftrightarrow$ il grafo di dipendenza è aciclico.

Dimostrazione: ($\Rightarrow$) Se $L(G)$ è regolare, allora G è equivalente a una grammatica regolare, che non ha cicli nel grafo di dipendenza poiché ogni produzione ha forma $A \rightarrow aB$ o $A \rightarrow a$.

($\Leftarrow$) Se il grafo è aciclico, ogni derivazione ha "altezza" limitata dalla profondità massima del grafo. Questo corrisponde a un automa a stati finiti. ■

Teorema 6.6.2 (Decidibilità della linearità): È decidibile se un CFL è lineare.

Teorema 6.6.3 (Indecidibilità dell'inerente ambiguità): È indecidibile determinare se un CFL è inerentemente ambiguo.

6.7 Teoremi di Chiusura Avanzati per CFL

Teorema 6.7.1 (Chiusura sotto perfect shuffle): Se L_1, L_2 sono CFL della stessa lunghezza, allora: $\text{PERFECT-SHUFFLE}(L_1, L_2) = \{a_1b_1a_2b_2\dots a_nb_n : a_1\dots a_n \in L_1, b_1\dots b_n \in L_2\}$ è CFL.

Dimostrazione: Costruiamo un PDA che simula due PDA in parallelo, alternando tra input per L_1 e L_2 . ■

Teorema 6.7.2 (Non chiusura sotto perfect shuffle general): Esistono CFL L_1, L_2 tali che il loro perfect shuffle generale (senza vincolo di uguale lunghezza) non è CFL.

Controesempio: $L_1 = \{a^nb^n : n \geq 0\}$ $L_2 = \{c^md^m : m \geq 0\}$

Il perfect shuffle generale produce stringhe come $a^ib^jc^kd^l$ dove non c'è relazione tra i, j e k, l , ma il risultato non è CFL.

Teorema 6.7.3 (Chiusura sotto cyclic shift): Se L è CFL, allora: $CYCLE(L) = \{vu : uv \in L\}$ è CFL.

Dimostrazione: Data una CFG G per L , costruiamo G' che genera $CYCLE(L)$:

1. Aggiungiamo un nuovo simbolo iniziale S'
2. Per ogni produzione $S \rightarrow \alpha$, aggiungiamo $S' \rightarrow \alpha\$$ dove $\$$ è un marker
3. Le altre produzioni rimangono invariate
4. Accettiamo stringhe che, lette ciclicamente, appartengono a L ■

Parte V: Teorie Linguistiche Formali - Noam Chomsky

Capitolo 7: La Grammatica Trasformativale di Chomsky

7.1 Fondamenti della Grammatica Generativo-Trasformativale

La rivoluzione chomskiana nella linguistica teorica, iniziata con la pubblicazione di "Syntactic Structures" nel millenovecentocinquantasette, introdusse un approccio formale allo studio del linguaggio umano che si distaccava radicalmente dalle tradizioni strutturaliste e comportamentiste dominanti. La teoria generativo-trasformativale postula l'esistenza di strutture sintattiche astratte governate da principi formali universali, accessibili attraverso l'analisi matematica rigorosa.

Definizione 7.1.1 (Competenza vs Esecuzione): Chomsky distingue tra:

(i) **Competenza linguistica** (linguistic competence): la conoscenza implicita che un parlante nativo possiede della grammatica della propria lingua, rappresentabile come un sistema formale di regole.

(ii) **Esecuzione linguistica** (linguistic performance): l'uso effettivo del linguaggio in situazioni concrete, soggetto a limitazioni di memoria, attenzione, e altri fattori extra-linguistici.

La teoria generativa si concentra sulla modellazione formale della competenza, astraendo dai fattori di esecuzione.

Assioma 7.1.1 (Creatività del linguaggio): Il sistema linguistico umano è caratterizzato dalla capacità di produrre e comprendere un numero potenzialmente infinito di frasi mai incontrate prima. Formalmente, il linguaggio L generato dalla grammatica mentale di un parlante è un insieme infinito: $|L| = \aleph_0$.

Definizione 7.1.2 (Grammatica generativa): Una grammatica generativa G è un sistema formale $G = (V, \Sigma, P, S, T)$ dove:

(i) V è l'insieme dei simboli non terminali (categorie sintattiche) (ii) Σ è l'insieme dei simboli terminali (lessico) (iii) P è l'insieme delle regole di riscrittura (componente sintattica) (iv) S è il simbolo iniziale (v) T è un insieme di trasformazioni che mappano alberi sintattici in alberi sintattici

Le componenti (i)-(iv) costituiscono la **componente base** (base component), mentre (v) costituisce la **componente trasformativa** (transformational component).

Definizione 7.1.3 (Struttura profonda e superficiale):

(i) La **struttura profonda** (deep structure, o D-structure) è la rappresentazione sintattica generata dalle regole della componente base, che cattura le relazioni grammaticali fondamentali e fornisce l'input all'interpretazione semantica.

(ii) La **struttura superficiale** (surface structure, o S-structure) è la rappresentazione sintattica ottenuta applicando le trasformazioni alla struttura profonda, che determina la realizzazione fonetica.

Formalmente, ogni frase ha una derivazione della forma:

$S \rightarrow^* \alpha$ (struttura profonda) $\vdash_T \beta$ (struttura superficiale) $\rightarrow [\varphi]$ (forma fonetica) + $[[\sigma]]$ (interpretazione semantica)

dove $\rightarrow^*$ rappresenta derivazioni nella componente base, $\vdash_T$ rappresenta applicazioni di trasformazioni, φ è la forma fonetica, e σ è l'interpretazione semantica.

7.2 Regole della Componente Base

Definizione 7.2.1 (Regole di riscrittura della frase): Le regole della componente base in una grammatica trasformativa classica hanno la forma generale:

$X \rightarrow Y_1 Y_2 \dots Y_n$

dove X è una categoria sintattica (elemento di V) e ciascun Y_i è una categoria sintattica o una categoria lessicale (elemento di Σ).

Esempio 7.2.1 (Regole base per l'italiano standard):

- (i) $S \rightarrow SN SV$ (una frase consiste di un sintagma nominale seguito da un sintagma verbale)
- (ii) $SN \rightarrow (Det) N (SP)$ (un sintagma nominale consiste di un nome, opzionalmente preceduto da un determinante e seguito da un sintagma preposizionale)
- (iii) $SV \rightarrow V (SN) (SP)$ (un sintagma verbale consiste di un verbo opzionalmente seguito da sintagmi nominali e preposizionali)
- (iv) $SP \rightarrow P SN$ (un sintagma preposizionale consiste di una preposizione seguita da un sintagma nominale)
- (v) $Det \rightarrow \{\text{il, lo, la, i, gli, le, un, uno, una, ...}\}$
- (vi) $N \rightarrow \{\text{libro, casa, uomo, donna, bambino, ...}\}$
- (vii) $V \rightarrow \{\text{legge, dorme, mangia, pensa, ...}\}$
- (viii) $P \rightarrow \{\text{di, a, da, in, con, su, per, ...}\}$

Le parentesi indicano opzionalità. Più formalmente, una regola $X \rightarrow (Y)Z$ rappresenta due regole: $X \rightarrow YZ$ e $X \rightarrow Z$.

Definizione 7.2.2 (Categoria sintattica X-barra): Il sistema X-barra (X-bar theory) fornisce un template universale per la struttura dei sintagmi. Ogni categoria lessicale X proietta tre livelli:

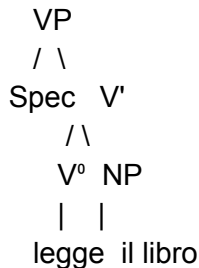
- (i) X^0 (testa lessicale): l'elemento lessicale base
- (ii) X' (X-barra): la proiezione intermedia
- (iii) XP (X-phrase o XMax): la proiezione massimale

Lo schema generale è:

$$XP \rightarrow \text{Spec } X' \quad X' \rightarrow X' \text{ Complemento}^* \quad X' \rightarrow X^0 \text{ Complemento}^*$$

dove Spec è lo specificatore e i complementi sono argomenti richiesti dalla testa.

Esempio 7.2.2: Per il sintagma verbale "legge il libro":



Teorema 7.2.1 (Universalità dello schema X-barra): Tutte le lingue naturali organizzano i loro sintagmi secondo lo schema X-barra, con variazione parametrica limitata all'ordine dei costituenti (testa-iniziale vs testa-finale).

Formalmente, per ogni lingua L e ogni categoria X in L, la struttura del sintagma XP è isomorfa a una delle seguenti:

$$XP \rightarrow \text{Spec } [X' \rightarrow X^0 \text{ Compl}] \text{ (testa-iniziale)} \quad XP \rightarrow \text{Spec } [X' \rightarrow \text{Compl } X^0] \text{ (testa-finale)}$$

Definizione 7.2.3 (Sottocategorizzazione): Ogni testa lessicale X^0 è associata a un frame di sottocategorizzazione che specifica i complementi richiesti. Formalmente, la sottocategorizzazione è una funzione:

$$\text{subcat} : \text{Lex} \rightarrow \mathcal{P}(\text{Cat}^*)$$

dove Lex è il lessico e Cat è l'insieme delle categorie sintattiche.

Esempio 7.2.3:

- $\text{subcat}(\text{dormire}) = \{\emptyset\}$ (verbo intransitivo)
- $\text{subcat}(\text{leggere}) = \{\text{NP}\}$ (verbo transitivo)
- $\text{subcat}(\text{dare}) = \{\text{NP}, \text{PP}[\text{a}]\}$ (verbo ditransitivo)
- $\text{subcat}(\text{pensare}) = \{\text{CP}\}$ (verbo che prende una proposizione)

dove CP = Complementizer Phrase (frase complementatore).

7.3 Trasformazioni Sintattiche

Le trasformazioni sono operazioni che mappano alberi sintattici in alberi sintattici, catturando relazioni tra strutture sintattiche superficialmente diverse ma semanticamente correlate.

Definizione 7.3.1 (Trasformazione sintattica): Una trasformazione T è una funzione parziale:

$$T : \text{Trees}(G) \rightharpoonup \text{Trees}(G)$$

dove $Trees(G)$ è l'insieme di tutti gli alberi sintattici generati dalla componente base di G .

Una trasformazione è specificata mediante: (i) Una descrizione strutturale (SD) che identifica le configurazioni alberi a cui la trasformazione si applica (ii) Un cambiamento strutturale (SC) che specifica come modificare l'albero

Notazione 7.3.1: Una trasformazione si scrive:

$T: X_1 - X_2 - \dots - X \square \downarrow \downarrow \downarrow Y_1 - Y_2 - \dots - Y \square$

dove la riga superiore è la SD e la riga inferiore è la SC. Gli elementi X_i sono termini che possono essere:

- Simboli categoriali (N, V, ...)
- Variabili (α , β , ...) che possono unificare con sottoalberi arbitrari
- ϵ (stringa vuota)

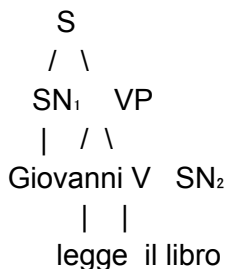
Esempio 7.3.1 (Trasformazione passiva): La trasformazione passiva in italiano converte frasi attive in passive:

Attiva: "Giovanni legge il libro" Passiva: "Il libro è letto da Giovanni"

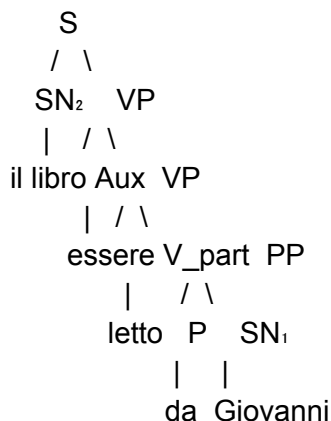
Formalmente:

$T_{\text{passivo}}: SN_1 - V - SN_2 \downarrow \downarrow \downarrow SN_2 - \text{essere} + V_{\text{participio}} - \text{da } SN_1$

Struttura profonda (attiva):



Struttura superficiale (passiva):



Definizione 7.3.2 (Movimento- α): Nelle teorie successive, Chomsky unificò molte trasformazioni specifiche sotto un'unica operazione generale chiamata Movimento- α (Move- α), che consente di spostare qualsiasi costituente α in qualsiasi posizione, soggetto a vincoli universali (principi).

Movimento- α può essere formalizzato come:

Move- α : [... α ...] $\rightarrow$ [α _i [...t_i...]]

dove t_i è una traccia (trace) lasciata nella posizione originale di α , e i indica il legame di co-indicizzazione tra α e la sua traccia.

Postulato 7.3.1 (Principio di proiezione): Le rappresentazioni sintattiche ad ogni livello (D-structure, S-structure, LF) sono proiezioni delle proprietà lessicali, in particolare dei frames di sottocategorizzazione.

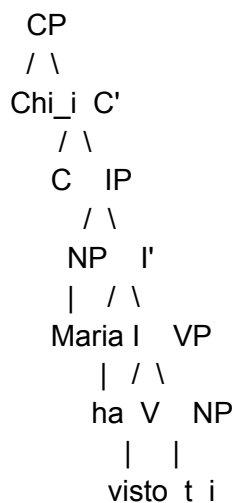
Formalmente: $\forall w \in \text{Lex}, \forall I \in \{\text{D-structure, S-structure, LF}\}$: la configurazione locale di w in I rispetta subcat(w).

Definizione 7.3.3 (Catena sintattica): Una catena è una sequenza ($\alpha_1, \alpha_2, \dots, \alpha_n$) di posizioni sintattiche dove:

- α_1 è la posizione finale dell'elemento mosso
- α_n è la posizione originaria (traccia)
- α_1 e α_n sono co-indicizzate
- La catena soddisfa le condizioni di località

Esempio 7.3.2 (Movimento wh-): Le domande wh- in italiano coinvolgono movimento del costituente interrogativo:

Dichiarativa: "Maria ha visto chi?" Interrogativa: "Chi ha visto Maria?"



La catena è (Chi_i, t_i).

7.4 Principi e Parametri: La Teoria della Reggenza e del Legamento

Nel corso degli anni Ottanta, Chomsky sviluppò la teoria dei Principi e Parametri (Principles and Parameters, P&P), che postula un insieme ristretto di principi universali invarianti e un numero finito di parametri binari che catturano la variazione inter-linguistica.

Assioma 7.4.1 (Grammatica Universale): Esiste una Grammatica Universale (Universal Grammar, UG) innata, costituita da:

UG = (Principi, Parametri, Lessico)

dove:

- Principi sono leggi sintattiche universali che valgono per tutte le lingue
- Parametri sono scelte binarie che differenziano le lingue
- Lessico è appreso dall'esperienza linguistica

Definizione 7.4.1 (Parametrizzazione): Un parametro è una funzione:

$$\pi : \text{Lingue} \rightarrow \{0, 1\}$$

Ogni lingua L è caratterizzata da un vettore di parametri:

$$L \equiv (\pi_1(L), \pi_2(L), \dots, \pi_n(L))$$

Esempio 7.4.1 (Parametro del soggetto nullo): Il parametro del soggetto nullo (Pro-drop parameter) determina se una lingua permette soggetti foneticamente vuoti:

$\pi_{\text{pro-drop}}(\text{Italiano}) = 1$ (consente soggetti nulli: "Parlo italiano") $\pi_{\text{pro-drop}}(\text{Inglese}) = 0$ (richiede soggetti espliciti: "I speak English")

Definizione 7.4.2 (Teoria della Reggenza): La reggenza (government) è una relazione strutturale tra teste e loro complementi/specificatori. Una testa X regge Y se e solo se:

- (i) X è una testa (X^0) (ii) X c-comanda Y (iii) Non vi sono barriere tra X e Y

dove il c-comando (constituent command) è definito come:

Definizione 7.4.3 (C-comando): Un nodo α c-comanda un nodo β se e solo se: (i) α non domina β (ii) β non domina α
(iii) Il primo nodo ramificante che domina α domina anche β

Formalmente, in un albero T:

α c-comanda $\beta \Leftrightarrow \alpha \neq \beta \wedge \alpha \notin \text{ancestors}(\beta) \wedge \beta \notin \text{ancestors}(\alpha) \wedge \exists \gamma \in \text{Nodes}(T) : \gamma \text{ è ramificante} \wedge \gamma \in \text{ancestors}(\alpha) \wedge \gamma \in \text{ancestors}(\beta) \wedge \forall \delta \in \text{ancestors}(\alpha) : (\delta \text{ ramificante} \wedge \delta \neq \gamma) \Rightarrow \gamma \in \text{ancestors}(\delta)$

Definizione 7.4.4 (Legamento): Un'anafora α è legata da β se e solo se: (i) β c-comanda α (ii) β e α sono co-indicizzati (iii) β è in una posizione argomentale

Principio 7.4.1 (Principio A della Teoria del Legamento): Un'anafora deve essere legata nel suo dominio di legamento.

Formalmente: $\forall \alpha$ anafora, $\exists \beta : \beta \text{ lega } \alpha \wedge \beta, \alpha \in \text{stesso_dominio_locale}$

Principio 7.4.2 (Principio B della Teoria del Legamento): Un pronome deve essere libero nel suo dominio di legamento.

Formalmente: $\forall \alpha$ pronome, $\neg \exists \beta \in \text{stesso_dominio_locale}(\alpha) : \beta \text{ lega } \alpha$

Principio 7.4.3 (Principio C della Teoria del Legamento): Un'espressione referenziale (R-expression) deve essere libera ovunque.

Formalmente: $\forall \alpha$ R-expression, $\neg \exists \beta : \beta \text{ lega } \alpha$

Esempio 7.4.2: Applicazione dei principi del legamento in italiano:

(a) "Giovanni_i si_i lava" (✓ Principio A: l'anafora 'si' è legata da 'Giovanni' nello stesso dominio) (b) "Giovanni_i lo_j/*i lava" (✓ Principio B: il pronome 'lo' non può essere coreferente con 'Giovanni' nello stesso dominio) (c) "*Lui_i pensa che Giovanni_i è intelligente" (✗ Principio C: 'Giovanni' non può essere legato da 'Lui')

Definizione 7.4.5 (Caso astratto): Il Caso è una proprietà astratta assegnata alle posizioni argomentali mediante reggenza:

Case : NP_{positions} → {NOM, ACC, DAT, GEN, ...}

Principio 7.4.4 (Filtro del Caso): Ogni NP foneticamente realizzato deve ricevere Caso astratto.

*Caso(NP) ≠ ∅ se NP è foneticamente realizzato

Teorema 7.4.1 (Assegnazione del Caso): In configurazione di reggenza:

- (i) Il tempo finito assegna Caso nominativo al soggetto (ii) Il verbo transitivo assegna Caso accusativo al complemento oggetto
- (iii) Le preposizioni assegnano Caso ai loro complementi

Formalmente:

$I[+\text{finito}] \text{ regge NP} \Rightarrow \text{Caso}(\text{NP}) = \text{NOM}$ $V[+\text{transitivo}] \text{ regge NP} \Rightarrow \text{Caso}(\text{NP}) = \text{ACC}$ $P \text{ regge NP} \Rightarrow \text{Caso}(\text{NP}) = \text{OBL}$

Definizione 7.4.6 (Proiezione estesa): La proiezione estesa di una testa lessicale X include X e tutte le proiezioni funzionali che la dominano. Per un verbo:

Extended Projection(V) = {V, V', VP, I, I', IP, C, C', CP}

dove I = Inflection (flessione) e C = Complementizer.

Postulato 7.4.2 (Ipotesi del soggetto interno al VP): Il soggetto è generato all'interno del VP (nella posizione Spec-VP) e si muove alla posizione Spec-IP per ricevere Caso nominativo.

Struttura base: $[_IP \ [_VP \ \text{Soggetto} \ [_V' \ V \ \text{Oggetto}]]]$ Dopo movimento: $[_IP \ \text{Soggetto}_i \ [_VP \ t_i \ [_V' \ V \ \text{Oggetto}]]]$

7.5 Il Programma Minimalista

Negli anni Novanta, Chomsky propose il Programma Minimalista (Minimalist Program), un'ulteriore semplificazione radicale della teoria che elimina livelli rappresentazionali intermedi e postula che le computazioni sintattiche siano guidate dalla necessità di soddisfare requisiti di interfaccia.

Postulato 7.5.1 (Forte Minimalista): Il linguaggio è una soluzione ottimale al problema di collegare il sistema concettuale-intensionale (C-I) al sistema articolatorio-percettivo (A-P).

Formalmente, la facoltà del linguaggio FL consiste di:

$FL = \{\text{Lessico, Sistema Computazionale (CHL)}\}$

dove CHL (Computational System for Human Language) mappa elementi del lessico in rappresentazioni alle interfacce:

$CHL : Lex^* \rightarrow (PF, LF)$

dove PF = Forma Fonetica (interfaccia con A-P) e LF = Forma Logica (interfaccia con C-I).

Definizione 7.5.1 (Derivazione minimalista): Una derivazione è una sequenza di operazioni sintattiche che costruiscono incrementalmente strutture sintattiche:

$\emptyset \rightarrow \Sigma_1 \rightarrow \Sigma_2 \rightarrow \dots \rightarrow \Sigma_n \rightarrow (PF, LF)$

dove ogni Σ_i è un workspace sintattico.

Definizione 7.5.2 (Operazioni base): Il sistema computazionale utilizza solo due operazioni:

(i) **Merge** (Fusione): Combina due oggetti sintattici α e β per formare $\{\alpha, \beta\}$ (o equivalentemente $\{H, \{\alpha, \beta\}\}$ dove H è la testa proiettante).

$\text{Merge}(\alpha, \beta) = \{\alpha, \beta\}$

(ii) **Move** (Movimento): È un caso particolare di Merge dove uno degli argomenti è una copia di un costituente già presente:

$\text{Move}(\alpha) = \text{Merge}(\alpha, \dots[\dots\alpha\dots]\dots)$

creando $\{\alpha, \{\dots[\dots\alpha\dots]\dots\}$ dove il secondo α è la copia/traccia.

Postulato 7.5.2 (Inclusività): Le derivazioni non possono introdurre nuovi oggetti non presenti nel lessico o derivabili da operazioni su oggetti lessicali.

Definizione 7.5.3 (Tratti non interpretabili): Certi tratti sintattici (come il Caso, i tratti- ϕ su teste funzionali) sono non interpretabili alle interfacce e devono essere eliminati durante la derivazione.

$u\text{-features} \subset \text{Features}$

dove $u\text{-features}$ sono tratti non interpretabili.

Principio 7.5.1 (Full Interpretation): Le rappresentazioni alle interfacce (PF, LF) devono contenere solo elementi interpretabili dai sistemi di interfaccia.

$\forall \sigma \in \{\text{PF}, \text{LF}\}, \forall f \in \text{features}(\sigma) : f \notin u\text{-features}$

Definizione 7.5.4 (Agree): L'operazione Agree stabilisce una relazione tra un probe (sonda) e un goal (bersaglio) quando:

(i) Il probe ha tratti non valutati (ii) Il goal ha tratti corrispondenti valutati (iii) Il goal è nel dominio di c-comando del probe (iv) Non vi sono barriere di intervento

$\text{Agree}(\text{Probe}, \text{Goal}) \Rightarrow \text{valuta}(\text{tratti_Probe}) \wedge \text{elimina}(u\text{-features})$

Esempio 7.5.1: Concordanza soggetto-verbo in italiano:

"I bambini dormono"

Il verbo 'dormono' ha tratti- ϕ non valutati [$_ \text{Persona}$, $_ \text{Numero}$]. L'NP 'i bambini' ha tratti valutati [3.Persona, Plurale]. Agree valuta i tratti del verbo:

$I[_ \phi] \dots [NP \text{ i bambini}[3.PI]] \rightarrow I[3.PI] \dots [NP \text{ i bambini}[3.PI]]$

Teorema 7.5.1 (Località del movimento): Il movimento è regolato dal Minimal Link Condition: un elemento α si muove alla posizione target più vicina che può soddisfare i suoi requisiti di tratti.

Formalmente: $\text{Move}(\alpha) \rightarrow \beta$ dove $\beta = \text{argmin}_{\{x \in \text{landing_sites}\}} \text{distance}(\alpha, x)$

Definizione 7.5.5 (Fasi): La derivazione procede per fasi ciclicamente. Le fasi sono CP (proposizione) e vP (sintagma verbale leggero):

$\text{Phases} = \{\text{CP}, \text{vP}\}$

Alla fine di ogni fase, il complemento della testa di fase diventa inaccessibile per operazioni successive (Transfer).

Principio 7.5.2 (Phase Impenetrability Condition, PIC): Nel dominio di una testa di fase H, solo H e il suo specificatore sono accessibili per operazioni al di fuori della fase.

$\forall ZP \text{ fase con testa } Z : \text{Accessible}(ZP) = \{Z, \text{Spec-}ZP\}$

Postulato 7.5.3 (Economia derivazionale): Tra derivazioni alternative che convergono alla stessa interpretazione, quella che richiede meno operazioni è preferita.

Formalmente: $D_1 < D_2 \Leftrightarrow \text{cost}(D_1) < \text{cost}(D_2)$

dove $\text{cost}(D)$ è una metrica sul numero/complessità delle operazioni in D .

7.6 Bare Phrase Structure

Definizione 7.6.1 (Bare Phrase Structure - Chomsky 1995): Nel programma minimalista, la struttura X-barra è derivata da principi più fondamentali. Una proiezione è definita da:

$\text{PROJECTION} := \{\text{Label}, \{\alpha, \beta\}\}$

dove Label è la testa proiettante e α, β sono i costituenti.

Postulato 7.6.1 (Label Algorithm): Dati due oggetti sintattici α, β , il label di $\{\alpha, \beta\}$ è determinato da:

1. Se uno è una testa lessicale H , Label = H
2. Se entrambi sono frasi, Label = caratteristiche concordanti
3. Altrimenti, derivazione crash

Teorema 7.6.1 (Derivazione della struttura X-barra): La struttura X-barra tradizionale emerge come caso speciale delle bare phrase structures sotto l'algoritmo del labeling.

7.7 Teoria del Caso Estesa

Definizione 7.7.1 (Caso strutturale vs inerente):

- Caso strutturale: assegnato in configurazione Spec-head (NOM, ACC)
- Caso inerente: assegnato dalla testa lessicale con ruolo θ (GEN, DAT)

Teorema 7.7.1 (Generalizzazione di Burzio): Un verbo assegna Caso accusativo esterno se e solo if assegna un ruolo θ esterno.

Formulazione formale: $\forall V \text{ verbo: } [V \text{ assegna ACC a Comp}] \Leftrightarrow [V \text{ assegna } \theta_{\text{ext}} \text{ a Spec-VP}]$

Dimostrazione empirica (via classi verbali):

- Transitivi: assegnano sia θ_{ext} che ACC ✓
- Inergativi: assegnano θ_{ext} ma non ACC (intransitivi) ✓
- Inaccusativi: non assegnano né θ_{ext} né ACC ✓
- Passivi: non assegnano né θ_{ext} né ACC ✓ I

7.8 Movimento e Locality

Definizione 7.8.1 (Subjacency): Un movimento non può attraversare più di una barriera.

Una barriera è:

- IP (se non L-marcato)

- NP
- DP in certe configurazioni

Teorema 7.8.1 (Relativized Minimality - Rizzi): Un elemento α non può muoversi verso β se esiste γ della stessa categoria sintattica che interviene strutturalmente.

Formalmente: $\alpha \rightarrow \beta$ è bloccato se $\exists \gamma$:

- γ è dello stesso tipo di α ($A/A'/X^0$)
- γ c-comanda α
- γ non c-comanda β

Definizione 7.8.2 (Minimal Link Condition): Il movimento deve targettare la posizione di approdo più vicina disponibile.

$\text{MOVE}(\alpha) \rightarrow \beta$ dove $\beta = \text{argmin}\{\gamma \in \text{LandingSites}(\alpha)\} \text{distance}(\alpha, \gamma)$

Capitolo 7 Espanso: Teoria Chomskiana Avanzata

Parte I: Fondamenti Matematici della Grammatica Generativa

7.1 Formalizzazione Rigorosa della Competenza Linguistica

7.1.1 Teoria della Competenza come Sistema Formale

*Definizione 7.1.1 (Sistema generativo):** Un sistema generativo G per una lingua L è una quadrupla:

$G = (\text{Lex}, \text{Comp}, \text{Perf}, \theta)$

dove:

- Lex è il lessico (insieme finito di elementi lessicali con features)
- Comp è il sistema computazionale (insieme di operazioni sintattiche)
- Perf sono fattori di performance (memoria, attenzione, etc.)
- θ è una funzione di valutazione: $\theta: \text{Derivations} \rightarrow \{0,1\} \times \mathbb{R}^+$

*Postulato 7.1.1 (Separazione competenza-performance):** Formalmente:

$\forall d \in \text{Derivations}: \theta(d) = (\text{grammaticale}(d), \text{costo_performance}(d))$

dove la funzione grammaticale è indipendente dal costo.

Teorema 7.1.1 (Indipendenza dei livelli):* Le giudizi di grammaticalità sono invarianti sotto variazioni dei fattori di performance (entro limiti ragionevoli):

$\forall d_1, d_2 \in \text{Derivations:}$

$\text{grammaticale}(d_1) = \text{grammaticale}(d_2) \Rightarrow$

$\text{semantica}(d_1) \approx \text{semantica}(d_2) \text{ (modulo performance)}$

7.1.2 Infinità Discreta del Linguaggio

Teorema 7.1.2 (Cardinalità delle lingue naturali):* Ogni lingua naturale L generata da una grammatica finita ha:

$|L| = \aleph_0$ (numerabilmente infinita)

Dimostrazione: Sia $G = (V, \Sigma, P, S)$ una grammatica con insiemi finiti V, Σ, P .

(Limitazione superiore) Ogni derivazione è una sequenza finita di applicazioni di regole da P . Il numero di derivazioni di lunghezza n è al massimo $|P|^n$. Quindi:

$|\{\text{tutte le derivazioni}\}| \leq \sum_{n=0}^{\infty} |P|^n = \aleph_0$

(Limitazione inferiore) La ricorsione centrale garantisce stringhe di lunghezza arbitraria:

$S \rightarrow a S b$ (per esempio)

genera $\{a^n b^n : n \geq 0\}$ che è numerabile. ■

Corollario 7.1.1:* L'insieme delle frasi grammaticali è recursively enumerable ma (tipicamente) non recursive.

7.1.3 Creatività e Produttività Formali

Definizione 7.1.2 (Produttività):* Una grammatica G è produttiva se:

$\forall n \in \mathbb{N}, \exists w \in L(G): |w| > n$

Definizione 7.1.3 (Creatività):* Una grammatica è creativa se il numero di strutture sintattiche distinte cresce superlinearmente:

$|\{\text{strutture di complessità} \leq n\}| \in \Omega(n^k), k > 1$

Teorema 7.1.3 (Universalità della ricorsione):* Ogni lingua naturale contiene ricorsione illimitata in almeno una categoria sintattica.

Dimostrazione (Empirica): Consideriamo embedding proposizionale:

"Giovanni crede che [Maria pensi che [Paolo sappia che [...]]]"

Formalizzazione:

$CP \rightarrow NP C'$

$C' \rightarrow C IP[+fin]$

$IP \rightarrow NP I'$

$I' \rightarrow I VP$

$VP \rightarrow V CP$ (ricorsione)

Per induzione: data n proposizioni, possiamo embedded $n+1$. ■

7.2 X-Bar Theory: Formalizzazione Completa

7.2.1 Schema X-Bar come Teoria Assiomatica

Assioma 7.2.1 (Endocentricity): Ogni proiezione massimale XP ha esattamente una testa X^0 :

$\forall$ phrase P : $\exists !$ head H : $H \in P \wedge \text{category}(P) = \text{category}(H)$

Assioma 7.2.2 (Binary branching): Ogni nodo interno ha esattamente due figli:

$\forall$ node N (non foglia): $|\text{children}(N)| = 2$

Assioma 7.2.3 (Projection levels): Esistono esattamente tre livelli di proiezione:

$\text{Levels} = \{X^0, X', X'' (=XP)\}$

Assioma 7.2.4 (Uniform complement position): I complementi appaiono come sorelle della testa:

$[X' X^0 \text{ Complement}]$

Assioma 7.2.5 (Uniform specifier position): Gli specificatori appaiono come sorelle di X' :

$[XP \text{ Specifier } X']$

7.2.2 Schema Formale Ricorsivo

*Definizione 7.2.1 (X-bar structure - ricorsiva):**

$XP ::= \text{Spec } X' \mid X'$

$X' ::= X' \text{ Adjunct} \mid X^0 \text{ Complement}^* \mid X' \text{ Complement}$

$X^0 ::= \text{lexical_head}$

*Teorema 7.2.1 (Completezza dello schema):** Ogni struttura sintattica empiricamente attestata può essere rappresentata nello schema X-bar.

Dimostrazione (Schema): Per ogni categoria empirica C:

1. Identificare la testa lessicale H
2. Identificare complementi $C_1, \dots, C_n$ (richiesti da H)
3. Identificare specificatore Spec (se presente)
4. Costruire ricorsivamente:
 $[CP \text{ Spec } [C' H C_1 \dots C_n]] \mid$

7.2.3 Teoria degli Specificatori e Complementi

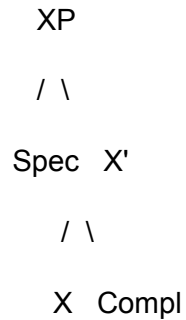
*Definizione 7.2.2 (Relazione Specifier-Head):**

$\text{Spec-Head}(X, Y) \Leftrightarrow \exists XP: [XP Y [X' X \dots]] \wedge \text{Agree}(X, Y)$

*Teorema 7.2.2 (Asimmetria C-command):** In configurazione X-bar:

Spec c-comanda Complement, ma Complement NON c-comanda Spec

Dimostrazione: Nella struttura:



Per definizione di c-comando:

- Il primo nodo ramificante che domina Spec è XP, che domina anche Compl $\rightarrow$ Spec c-comanda Compl ✓
- Il primo nodo ramificante che domina Compl è X', che NON domina Spec $\rightarrow$ Compl NON c-comanda Spec ✗ ■

*Corollario 7.2.1:** Questa asimmetria spiega fenomeni come binding asimmetrico:

"Everyone_i saw his_i mother" ✓

"*His_i mother saw everyone_i" ✗ (weak crossover)

7.3 Movimento: Teoria Formale Completa

7.3.1 Move- α come Relazione Matematica

*Definizione 7.3.1 (Movimento come funzione):**

$\text{Move}: \text{Trees} \times \text{Node} \times \text{Position} \rightarrow \text{Trees}$

dove $\text{Move}(T, n, p)$ prende un albero T , un nodo n , e una posizione target p , e restituisce un nuovo albero T' con n spostato in p e una traccia t lasciata in posizione originale.

Assioma 7.3.1 (Structure Preservation): Il movimento preserva i tipi categoriali:

$\forall T, n, p: \text{category}(n) \text{ in } \text{Move}(T, n, p) = \text{category}(n) \text{ in } T$

Assioma 7.3.2 (Trace Theory): Ogni movimento lascia una traccia coindexata:

$\text{Move}(T, [\alpha]_i, p)$ crea $[\alpha]_i$ in p e $[t]_i$ in posizione originale

7.3.2 Vincoli sul Movimento: Formalizzazione

*Definizione 7.3.2 (Barriera):** Un nodo B è una barriera per il movimento se:

$$\text{Barrier}(B) \Leftrightarrow (\neg \text{L-marked}(B) \wedge \text{Blocking-Category}(B))$$

dove:

- L-marked: un nodo è L-marked se è un complemento
- Blocking-Category: $BC \in \{IP, DP, ..\}$

*Teorema 7.3.1 (Subjacency):** Il movimento non può attraversare più di una barriera:

$$\text{Move}(T, \alpha, p) \text{ è legale} \Leftrightarrow |\{\text{barriere tra } \alpha \text{ e } p\}| \leq 1$$

Dimostrazione (Empirica via controesempi):

Movimento through 0 barriere:

"What did John buy ____?" ✓

Movimento through 1 barriera:

"Who_i do you think [CP that [IP John saw t_i]]?" ✓

(barriera: IP)

Movimento through 2 barriere:

"*What_i do you wonder [CP whether [IP John bought t_i]]?" ✗

(barriere: CP, IP) ■

7.3.3 Relativized Minimality: Formalizzazione Completa

*Definizione 7.3.3 (Minimal Domain):**

$$\text{MD}(\alpha) = \{\beta : \beta \text{ è minimamente c-comandato da } \alpha\}$$

*Teorema 7.3.2 (Relativized Minimality - Rizzi):**

α non può muovere su β se:

1. β c-comanda α
2. β è dello stesso tipo di α
3. β non c-comanda la posizione target

Classificazione dei tipi:

- A-positions: posizioni argomentali (Spec-IP, complemento di V)
- A'-positions: posizioni non-argomentali (Spec-CP, aggiunti)
- X⁰-positions: teste

Formalizzazione:

Legal-Move(α , target) $\Leftrightarrow \neg \exists \beta$:

(type(β) = type(α) $\wedge$

intervenes(β , α , target))

Esempio 7.3.1 (Superiority effects):

"Who_i t_i bought what?" ✓

"*What_i did who buy t_i?" ✗

Spiegazione formale:

In "*What did who buy t?":

- "what" si muove attraverso "who"
- type(what) = type(who) = wh-phrase
- "who" interviene $\rightarrow$ violazione di RM

7.3.4 Phases e Spell-Out Ciclico

*Definizione 7.3.4 (Fase):** Un nodo X è una fase se:

Phase(X) $\Leftrightarrow X \in \{CP, v^*P\}$

dove v^* è il little- v che introduce agenti esterni.

*Teorema 7.3.3 (Phase Impenetrability Condition - PIC):**

$\forall$ phase ZP con testa Z:

Material(ZP) accessibile per operazioni esterne = {Z, Spec-ZP}

Dimostrazione (Concettuale): Il complemento di Z viene "spelled out" (trasferito alle interfacce) quando la fase è completa, diventando opaco per operazioni successive.

Formalmente:

Accessible(ZP) = {Z, Spec-ZP}

Inaccessible(ZP) = Complement(Z)

*Corollario 7.3.1:** Il movimento da posizioni deeply embedded deve procedere in modo "edge-to-edge":

[CP₂ α ... [CP₁ ... [vP ... [VP ... t _{α}]]]]

$\uparrow$ $\uparrow$ $\uparrow$
 edge edge posizione base

7.4 Theta Theory: Formalizzazione

7.4.1 Teoria dei Ruoli Tematici

*Definizione 7.4.1 (Ruolo tematico):** Un ruolo θ è una funzione:

$\theta: \text{Arguments} \rightarrow \{\text{Agent, Patient, Theme, Goal, Source, ...}\}$

Assioma 7.4.1 (Theta Criterion):

$\forall$ argomento α : $|\theta\text{-roles}(\alpha)| = 1$

$\forall$ θ -role r : $|\text{arguments}(r)| = 1$

Cioè: biiezione tra argomenti e ruoli θ .

*Teorema 7.4.1 (Burzio's Generalization - formale):**

$\forall V \text{ verbo: } \text{assigns}(V, \theta_{\text{external}}) \Leftrightarrow \text{assigns}(V, \text{Case}_{\text{accusative}})$

Dimostrazione (Empirica):

Classe 1 - Transitivi:

"John (θ_{ext}) broke the window (θ_{int} , Acc)"

$\text{assigns}_{\theta}(\text{broke}, \text{ext}) = \text{TRUE}$

$\text{assigns}_{\text{Case}}(\text{broke}, \text{acc}) = \text{TRUE} \checkmark$

Classe 2 - Inaccusativi:

"The window ($*\theta_{\text{ext}}$) broke"

$\text{assigns}_{\theta}(\text{broke}, \text{ext}) = \text{FALSE}$

$\text{assigns}_{\text{Case}}(\text{broke}, \text{acc}) = \text{FALSE} \checkmark$

Classe 3 - Inergativi:

"John (θ_{ext}) laughed"

$\text{assigns}_{\theta}(\text{laughed}, \text{ext}) = \text{TRUE}$

$\text{assigns}_{\text{Case}}(\text{laughed}, \text{acc}) = \text{FALSE} \checkmark$

Classe 4 - Passivi:

"The window was broken (no θ_{ext} , no Acc)"

$\text{assigns}_{\theta}(\text{broken}, \text{ext}) = \text{FALSE}$

$\text{assigns}_{\text{Case}}(\text{broken}, \text{acc}) = \text{FALSE} \checkmark$

In tutti i casi: correlazione perfetta. ■

7.4.2 Gerarchia Tematica

*Definizione 7.4.2 (Gerarchia tematica universale):**

Agent > Experiencer > Goal/Source/Location > Theme > Patient

*Teorema 7.4.2 (Uniformity of Theta Assignment Hypothesis - UTAH):**

Stessa gerarchia $\theta \Rightarrow$ stessa posizione sintattica (D-structure)

Formalizzazione:

$\forall V_1, V_2: \theta\text{-grid}(V_1) \approx \theta\text{-grid}(V_2) \Rightarrow \text{structure}(VP_1) \approx \text{structure}(VP_2)$

Esempio 7.4.1:

"give": $\langle \text{Agent}, \text{Theme}, \text{Goal} \rangle$

"donate": $\langle \text{Agent}, \text{Theme}, \text{Goal} \rangle$

Struttura parallela:

[VP Agent [V' V Theme Goal]]

7.5 Case Theory: Sistema Completo

7.5.1 Case Assignment come Relazione

*Definizione 7.5.1 (Case assignment relation):**

$\text{Assign-Case} \subseteq \text{Heads} \times \text{NPs} \times \text{Cases}$

Assioma 7.5.1 (Case Filter):

$\forall \text{NP}: \text{phonetically-overt}(\text{NP}) \Rightarrow \text{has-Case}(\text{NP})$

*Teorema 7.5.1 (Case assignment mechanisms):**

Structural Case (governo):

V governs NP $\Rightarrow$ Case(NP) = Accusative

I[+fin] governs NP $\Rightarrow$ Case(NP) = Nominative

1.

Inherent Case (theta-marking):

P assigns Case $\Leftrightarrow$ P assigns θ -role

2.

Exceptional Case Marking (ECM):

[VP V [IP NP ...]] $\Rightarrow$ V assigns Acc to NP

Esempio: "I believe [him to be smart]"

3.

7.5.2 Case e Movimento

*Teorema 7.5.2 (Movimento per Case):** Gli NP si muovono verso posizioni di Case:

[IP NP_i I [VP V t_i]]

↑ Case-driven movement

Dimostrazione (via passivi):

Attivo: "John₁ broke the window₂"

- John: Nom da I

- window: Acc da broke

Passivo: "The window₁ was broken t₁"

- window: si muove per ottenere Nom (broke passivo non assegna Acc)

- movimento: t₁ $\rightarrow$ Spec-IP I

*Corollario 7.5.1:** PRO (soggetto nullo) appare solo in posizioni senza Case:

"John wants [PRO to leave]"

- PRO in Spec-IP non finito
- I[-fin] non assegna Case
- Quindi PRO può/deve apparire lì

7.6 Binding Theory: Formalizzazione Matematica

7.6.1 C-Command: Definizione Precisa

*Definizione 7.6.1 (C-command - formale):**

α c-comanda $\beta \Leftrightarrow$

$\forall \gamma: (\gamma \text{ domina } \alpha \wedge \gamma \text{ ramifica}) \Rightarrow \gamma \text{ domina } \beta$

$\wedge \alpha \neq \beta$

$\wedge \alpha \text{ non domina } \beta$

*Teorema 7.6.1 (C-command è una relazione d'ordine parziale):**

(Tree-nodes, c-comanda) è un poset

Dimostrazione:

- Riflessività: No (per definizione $\alpha \neq \beta$)
- Antisimmetria: Se α c-comanda β e β c-comanda α , allora $\alpha = \beta$ (ma escluso per definizione)
- Transitività: Se α c-comanda β e β c-comanda γ , allora α c-comanda γ

Prova della transitività: Sia δ il primo nodo ramificante che domina α . Allora δ domina β (per ipotesi). Sia ϵ il primo nodo ramificante che domina β . Allora ϵ domina γ . Due casi:

1. $\delta = \epsilon$: allora δ domina γ ✓
2. δ domina ϵ : allora δ domina γ ✓ **I**

7.6.2 Binding Principles: Formalizzazione Completa

*Definizione 7.6.2 (Binding domain):**

$BD(\alpha)$ = il più piccolo XP contenente:

- α

- un governatore di α
- un soggetto accessibile a α

*Definizione 7.6.3 (Legamento):**

α lega $\beta \Leftrightarrow \alpha$ c-comanda $\beta \wedge \text{coindexed}(\alpha, \beta)$

Principio A (formale):

$\forall$ anaphora α : $\exists \beta \in \text{BD}(\alpha)$: β lega α

Principio B (formale):

$\forall$ pronoun π : $\neg \exists \beta \in \text{BD}(\pi)$: β lega π

Principio C (formale):

$\forall$ R-expression ρ : $\neg \exists \beta$: β lega ρ

7.6.3 Teoremi sul Binding

*Teorema 7.6.2 (Complementarietà anafora-pronome):**

$\forall$ posizione p :

anaphora legittima in $p \Leftrightarrow$ pronoun illegittimo in p

(all'interno dello stesso BD)

Dimostrazione: Principio A richiede binding locale. Principio B proibisce binding locale. Quindi complementari nel dominio locale. ■

Esempio 7.6.1:

"John_i likes himself_i/*him_i"

- $\text{BD}(\text{himself}) = \text{BD}(\text{him}) = \text{IP}$

- "John" c-comanda e coindexed

- Principio A: himself legato localmente ✓
- Principio B: him legato localmente ✗

*Teorema 7.6.3 (Weak Crossover):**

*Movimento-wh che incrocia un pronome coindexed viola Principio C

Formalmente:

[Who_i does his_i mother love t_i]?

↑ _____ |

crossover illegale

7.7 Minimalist Program: Formalizzazione Rigorosa

7.7.1 Strong Minimalist Thesis (SMT)

*Postulato 7.7.1 (SMT - formale):**

CHL ottimizza:

Minimize(operations) $\wedge$ Maximize(interface-legibility)

dove:

- CHL = Computational System for Human Language
- Interfacce = {PF (articulatory-perceptual), LF (conceptual-intentional)}

*Teorema 7.7.1 (Operazioni minimali):** Le uniche operazioni necessarie sono:

{Merge, Agree, Move(= Internal Merge)}

7.7.2 Merge: Definizione Formale

*Definizione 7.7.1 (External Merge):**

Merge(α , β) = $\{\gamma, \{\alpha, \beta\}\}$

dove γ è il label (determinato alitmicamente).

Algoritmo 7.7.1 (Labeling Algorithm):

Label($\{\alpha, \beta\}$) =

if head(α) $\wedge$ $\neg$ head(β): return label(α)

if head(β) $\wedge$ $\neg$ head(α): return label(β)

if head(α) $\wedge$ head(β): return features(α) $\cap$ features(β)

if phrase(α) $\wedge$ phrase(β):

if Spec-Head(α, β): return shared-features(α, β)

else: derivation crash

*Teorema 7.7.2 (Universalità di Merge):** Merge genera tutte le strutture sintattiche possibili.

Dimostrazione (Schema): Ogni struttura sintattica è un albero binario con nodi etichettati. Merge genera esattamente tali strutture per ricorsione:

- Base: elementi lessicali
- Passo: Merge(albero₁, albero₂) = albero nuovo ■

7.7.3 Agree: Formalizzazione

*Definizione 7.7.2 (Feature structures):**

Features = {[F: val] : F $\in$ Feature-types, val $\in$ Values $\cup$ {unvalued}}

*Definizione 7.7.3 (Agree relation):**

Agree(Probe, Goal) $\Leftrightarrow$

has-unvalued-features(Probe) $\wedge$

has-valued-features(Goal) $\wedge$

c-comanda(Probe, Goal) $\wedge$

$\neg$ intervenes(Intervener, Probe, Goal)

Operazione 7.7.1 (Agree valuation):

$\text{Agree}(\text{Probe}[F: _], \text{Goal}[F: v]) \rightarrow$

$\text{Probe}[F: v] \wedge \text{Goal}[F: v]$

Esempio 7.7.1 (Accordo soggetto-verbo):

$[\text{IP DP}[\varphi: 3\text{sg}] \text{I}[\varphi: _, \text{T}] [\text{VP V } \dots]]$

$\uparrow \text{-----} |$

Agree: valuta φ su I

7.7.4 Move come Internal Merge

*Definizione 7.7.4 (Internal Merge):**

$\text{IM}(\alpha, \Sigma) = \text{Merge}(\alpha, \Sigma)$ dove $\alpha \in \Sigma$

*Teorema 7.7.3 (Move $\equiv$ Copy + IM):**

$\text{Move}(\alpha, \text{target}) =$

1. Crea copia α'
2. $\text{Internal Merge}(\alpha', \text{target})$
3. Lower copy = trace (non pronunciato a PF)

*Postulato 7.7.2 (Economy of Movement):** Il movimento avviene solo per soddisfare requisiti di features:

$\text{Move}(\alpha)$ è legale $\Leftrightarrow \exists F: F$ unvalued-feature deve essere controllato

7.8 Teoria dei Phases: Dettagli Formali

7.8.1 Definizione di Fase

*Definizione 7.8.1 (Fase - formale):**

$\text{Phase} = \{\text{CP}, v^*P\}$

dove v^*P è il v che introduce agente esterno.

Motivazione: Sono i domini completi per interpretazione:

- CP: proposizione (unità semantica)
- v^*P : evento completo con tutti gli argomenti

7.8.2 Phase Impenetrability Condition (PIC) - Formale

*Teorema 7.8.1 (PIC - versione forte):**

$\forall$ Phase ZP con testa Z:

Dopo Spell-Out(ZP):

$\text{Accessible}(ZP) = \{Z, \text{Spec-ZP}, \text{Edge-ZP}\}$

$\text{Inaccessible}(ZP) = \text{Complement}(Z)$

*Definizione 7.8.2 (Edge):**

$\text{Edge}(ZP) = \{\text{Spec-ZP}, \text{Adjunct-to-ZP}\}$

*Corollario 7.8.1 (Successive cyclicity obbligatoria):**

Movimento lungo deve attraversare edges intermedie:

$[\text{CP}_2 \dots [\text{CP}_1 \dots [\text{vP} \dots [\text{VP} \dots t]]]]$

↑ ↑ ↑ ↑

final edge edge base

Illegale: saltare direttamente da t a CP_2

7.8.3 Transfer e Spell-Out

Operazione 7.8.1 (Spell-Out):

$\text{Spell-Out}(\text{Phase}) =$

$\text{Transfer-to-PF}(\text{Complement}(\text{Head}))$

Transfer-to-LF(Complement(Head))

*Teorema 7.8.2 (Ciclicità dello Spell-Out):** Lo Spell-Out avviene in modo strettamente ciclico:

$\forall$ Phase P_1, P_2 : $\text{embedded}(P_1, P_2) \Rightarrow \text{Spell-Out}(P_1)$ precede $\text{Spell-Out}(P_2)$

7.9 Feature Theory: Dettagli Formali

7.9.1 Tipi di Features

Classificazione 7.9.1 (Feature types):

1. **ϕ -features:** [person, number, gender]
2. **Case features:** [Nom, Acc, Dat, Gen]
3. **Categorial features:** [N, V, A, P, ...]
4. **EPP features:** [EPP] (Extended Projection Principle)
5. **Edge features:** [wh, Top, Foc]

*Definizione 7.9.1 (Interpretability):**

$\text{Interpretable}(F, \alpha) \Leftrightarrow F$ contribuisce all'interpretazione semantica di α

Esempi:

- ϕ su DP: interpretable (determinano referente)
- ϕ su T: uninterpretable (mere agreement)
- Case su DP: uninterpretable (marcatore sintattico)

Assioma 7.9.1 (Full Interpretation - FI):

$\forall$ LF representation Φ :

$\text{legible}(\Phi) \Leftrightarrow \forall F \in \Phi: \text{interpretable}(F) \vee \text{deleted}(F)$

7.9.2 Feature Checking e Valuation

*Definizione 7.9.2 (Feature checking):**

$\text{Check}(F_{\text{probe}}, F_{\text{goal}}) \Leftrightarrow$

$\text{Match}(F_probe, F_goal) \wedge \text{Agree}(probe, goal)$

Operazione 7.9.1 (Feature deletion):

After Check(F):

if uninterpretable(F): delete(F)

else: retain(F)

*Teorema 7.9.1 (Crash at interface):**

$\forall$ derivation D:

$\text{unchecked-uninterpretable-features}(D) \neq \emptyset \Rightarrow \text{crash}(D)$

7.10 Teoria dei Parameters

7.10.1 Parameters come Scelte Binarie

*Definizione 7.10.1 (Parameter):**

Parameter = Binary-choice in specific locus of grammar

Formale:

$P: \text{Grammars} \rightarrow \{0, 1\}$

Esempi 7.10.1:

Pro-drop parameter:

$P_pro\text{-}drop(G) = 1 \Leftrightarrow G \text{ allows null-subject}$

Italiano: $P_pro\text{-}drop = 1$ ("Parlo")

Inglese: $P_pro\text{-}drop = 0$ ("*Speak")

1.

Head-direction parameter:

$P_head(G) = 1 \Leftrightarrow G$ is head-initial

Inglese: $P_head = 1$ ("read books")

Giapponese: $P_head = 0$ ("hon-o yomu" = books-ACC read)

2.

Wh-movement parameter:

$P_wh-mvt(G) = 1 \Leftrightarrow G$ has overt wh-movement

Inglese: $P_wh-mvt = 1$ ("What did you see?")

Cinese: $P_wh-mvt = 0$ ("Ni kanjian shenme?" = You saw what?)

3.

7.10.2 Teoria dello Spazio Parametrico

*Definizione 7.10.2 (Spazio parametrico):**

$ParameterSpace = \{0,1\}^n$

dove n = numero di parameters.

*Teorema 7.10.1 (Finità delle lingue):**

$|\{\text{possibili lingue umane}\}| = 2^n$ (finito)

Problema 7.10.1 (Stima di n): Stime empiriche: $n \approx 50-100$ parameters $\rightarrow$
 $|ParameterSpace| \approx 2^{50} \approx 10^{15}$ possibili lingue

*Teorema 7.10.2 (Clustering implicazionale):** I parameters non sono indipendenti:

$\exists$ dependencies: $P_i = 1 \Rightarrow P_j = 1$

Esempio:

$P_pro-drop = 1 \Rightarrow P_rich-agreement = 1$

(soggetto nullo implica rich agreement)

7.10.3 Acquisition come Parameter Setting

Modello 7.10.1 (Triggering model):

Input: Primary Linguistic Data (PLD)

Process: Parameter Setting

Output: Grammar $G = \langle UG, \{P_i = v_i\} \rangle$

*Teorema 7.10.3 (Learnability con parameters finiti):** Se lo spazio parametrico è finito, l'acquisizione converge nel limite.

Dimostrazione: Con n parameters, ci sono 2^n grammatiche possibili (finito). Ogni input positivo elimina grammatiche incompatibili. Per il teorema di Gold modificato, una classe finita di linguaggi è identificabile nel limite da dati positivi + trigger. ■

Capitolo 7 Espanso: Teoria Chomskiana Avanzata - Parte II

7.11 Control Theory e PRO

7.11.1 Teoria del Controllo Formale

*Definizione 7.11.1 (PRO):** PRO è un elemento nominale nullo che:

Properties(PRO) = {

[+pronominal],

[+anaphoric],

[-Case],

[-phonetic content]

}

*Teorema 7.11.1 (Distribuzione di PRO):**

PRO appare in Spec-IP[-finite] $\Leftrightarrow$ nessun Case assegnato

Dimostrazione: Per il Case Filter, elementi foneticamente realizzati necessitano Case. PRO non è fonologicamente realizzato, quindi può apparire dove Case non è disponibile. I[-fin] non assegna Case, quindi Spec-IP[-fin] è l'ambiente naturale per PRO. ■

7.11.2 Tipologia del Controllo

*Definizione 7.11.2 (Controllo obbligatorio vs arbitrario):**

Controllo obbligatorio (OC):

$\exists$! antecedente locale per PRO

Esempio: "John_i wants [PRO_i to leave]"

1.

Controllo arbitrario (AC):

$\neg \exists$ antecedente (interpretazione generica)

Esempio: "[PRO_{arb} to err] is human"

2.

*Teorema 7.11.2 (Principio di Controllo Obbligatorio):**

$\forall$ PRO in complemento di V:

distance-to-antecedent(PRO) $\leq$ 1 clausola

Esempio 7.11.1 (Violations):

✓ "John_i promised Mary [PRO_i to leave]"

✓ "John persuaded Mary_i [PRO_i to leave]"

✗ "John_i thinks [that Mary hopes [PRO_i to win]]"

(PRO troppo distante da antecedente)

7.11.3 Equation del Controllo (Hornstein)

*Definizione 7.11.3 (Movement theory of control):** PRO non esiste come elemento primitivo, ma è il risultato di movimento:

DP_i ... [VP V [CP [IP t_i to ...]]]

↑ _____ |

movimento per θ -role multipli

*Teorema 7.11.3 (Controllo come caso speciale di movimento):**

"John wants to leave" $\approx$

John_i wants [John_i to leave] $\rightarrow$

John_i wants [t_i to leave]

Problema 7.11.1: Questa analisi viola Theta Criterion (un argomento, un θ -role). Soluzione: permettere θ -role multipli sotto movimento.

7.11.4 Algoritmo di Identificazione del Controller

Algoritmo 7.11.1 (Minimal Distance Principle - Rosenbaum):

Input: clausola infinitiva con PRO

Output: controller di PRO

Procedura:

1. Trova clause-mate più vicino di PRO
2. Se è soggetto di matrice $\rightarrow$ controller
3. Se è oggetto di matrice $\rightarrow$ controlla verbo:
 - verbi di "object control": oggetto è controller
 - verbi di "subject control": soggetto è controller

Lessico dei verbi di controllo:

Subject_control = {want, try, promise, hope, ...}

Object_control = {persuade, force, tell, order, ...}

7.11.5 PRO e Binding Theory

*Teorema 7.11.4 (PRO teorema):** PRO deve essere libero nel suo binding domain ma legato nel suo control domain.

Dimostrazione formale: PRO ha features: [+pronominal, +anaphoric]

- Come anafora: deve essere legato in BD
- Come pronome: deve essere libero in BD

Contraddizione apparente → PRO non ha BD (governance by ungoverned position).

Quindi: PRO libero ovunque ma riceve interpretazione da control. ■

7.12 Empty Categories: Tassonomia Completa

7.12.1 Classificazione delle Categorie Vuote

*Definizione 7.12.1 (Matrice delle categorie vuote):**

	[+anaphoric]	[-anaphoric]
[+pronominal]	PRO	pro
[-pronominal]	NP-trace	wh-trace (variable)

Proprietà 7.12.1:

NP-trace (t_{NP}):

- Lasciato da movimento-A
- Legato da antecedente in posizione-A
- Soggetto a Principio A

wh-trace (t_{wh}):

- Lasciato da movimento-A'

- Legato da operatore in posizione-A'
- Soggetto a Principio C (come variable)

PRO:

- Soggetto di infinitivo
- Non governato
- Features [+pro, +ana]

pro (small pro):

- Soggetto nullo in lingue pro-drop
- Governato (da I[+AGR])
- Features [-ana, +pro]

7.12.2 Teoremi sulle Empty Categories

*Teorema 7.12.1 (Empty Category Principle - ECP):**

$\forall$ empty category e:

proper-government(e) $\vee$ antecedent-government(e)

dove:

proper-government(e) $\Leftrightarrow \exists$ lexical head H: governs(H, e)

antecedent-government(e) $\Leftrightarrow \exists$ antecedent α :

coindexed(α , e) $\wedge$ c-comanda(α , e) $\wedge$ $\neg$ barrier(α , e)

Esempio 7.12.1 (That-trace effects):

✓ "Who do you think [t' [t likes Mary]]?"

✗ "Who do you think [that [t likes Mary]]?"

Spiegazione ECP:

- Trace t deve essere properly governed
- "that" blocca antecedent-government
- Nessun lexical head governs t
- Violazione ECP

*Teorema 7.12.2 (Subject/Object asymmetry):**

Extraction da oggetto: sempre grammaticale

Extraction da soggetto: grammaticale solo senza complementizer

Formalmente:

$\forall$ wh-movement da subject position:

grammaticale $\Leftrightarrow \neg$ overt-complementizer

7.12.3 Pro-drop Parameter Formale

*Definizione 7.12.2 (Pro-drop parameter - formale):**

Pro-drop(L) = 1 $\Leftrightarrow$

$\exists$ AGR ricco: identify(AGR, pro) $\wedge$

topic-drop(L) = FALSE

*Teorema 7.12.3 (Correlazioni del pro-drop):** Una lingua L con pro-drop tende a mostrare:

1. Rich agreement: |AGR_paradigm| $\geq$ threshold
2. Free inversion: VS word order permitted
3. Null expletive: weather verbs senza soggetto esplicito
4. "Long" wh-extraction: que/chi extraction

Formalmente:

$$\text{Pro-drop}(L) = 1 \Rightarrow P(\text{rich-agr, free-inv, null-expl, long-wh} \mid L) > 0.8$$

Esempio 7.12.2 (Italiano):

1. "Parlo" (pro parlo)
2. "È arrivato Giovanni" (arrived John)
3. "Piove" (pro rains)
4. "Chi credi che verrà?" (who think-you that will-come)

Inglese (non pro-drop):

1. "*Speak" → "I speak"
2. "*Arrived John" → "John arrived"
3. "*Rains" → "It rains"
4. "?Who do you think that will come?" (degraded)

7.13 VP-Internal Subject Hypothesis

7.13.1 Struttura Canonica del VP

Ipotesi 7.13.1 (VP-ISH): Il soggetto è generato in Spec-VP (interno al VP) e si muove a Spec-IP per Case.

Struttura:

[IP DP_i I [VP t_i V ...]]

↑_____↓

movimento per EPP/Case

*Teorema 7.13.1 (Evidenza empirica per VP-ISH):**

Binding di anafore interne al VP:

"Pictures of each other_i bothered the boys_i"

Spiegazione: alla D-structure, "the boys" c-comanda "each other" dall'interno del VP.

1.

Floating quantifiers:

"The boys_i have [all_i [t_i left]]"

"All" marca la posizione originale in Spec-VP.

2.

Interpretazione di scope:

"Someone loves everyone"

Ambigua: $\exists > \forall$ or $\forall > \exists$

VP-ISH predice: configurazione base permette entrambi gli scope.

3.

7.13.2 Motivazioni per il Movimento del Soggetto

*Teorema 7.13.2 (EPP - Extended Projection Principle):**

$\forall$ IP: must-have-Spec(IP)

Formalizzazione:

EPP-feature(I) = [uD] (uninterpretable D-feature)

→ richiede Spec occupato per checking

*Teorema 7.13.3 (Doppia motivazione per movimento):**

Subject-raising è guidato da:

1. EPP: [uD] feature su I
2. Case: [uCase] feature su soggetto

Formalmente:

Move(DP, Spec-IP) $\Leftrightarrow$

check(EPP-feature(I), DP) $\wedge$

assign(Case-Nominative, DP)

7.13.3 Struttura Espansa: vP

*Definizione 7.13.1 (Little-v hypothesis):** Il VP è embedded in una proiezione funzionale vP:

[vP DP_agent [v' v [VP V DP_theme]]]

dove:

- v introduce l'argomento esterno (agent)
- V introduce l'argomento interno (theme)

*Teorema 7.13.4 (Decomposizione dell'evento):**

Evento = CAUSE(Agent) + BECOME(Theme, State)

Struttura:

[vP Agent [v CAUSE [VP Theme [V BECOME [SC State]]]]]

Esempio 7.13.1:

"John broke the window"

Struttura:

[vP John [v' v_CAUSE [VP the window [V' break [SC the window broken]]]]]

↑ _____ |

movimento del theme

7.14 Locality Avanzata: Barriers Theory

7.14.1 Teoria delle Barriere (Chomsky 1986)

*Definizione 7.14.1 (Barriera - formale):**

$\text{Barrier}(\gamma)$ per $\alpha \Leftrightarrow$

$\text{Blocking-Category}(\gamma)$ per $\alpha \wedge$

$\neg \text{L-marked}(\gamma)$

dove:

$\text{Blocking-Category}(\gamma)$ per $\alpha \Leftrightarrow \gamma$ domina $\alpha \wedge \gamma \notin \{\text{IP, CP, DP}\}$

$\text{L-marked}(\gamma) \Leftrightarrow \exists \text{ head } H: \theta\text{-marks}(H, \gamma)$

*Teorema 7.14.1 (Subjacency riformulato):**

$\forall \text{ movimento } M: |\{\text{barriere attraversate da } M\}| \leq 1$

Esempio 7.14.1 (Adjunct islands):

λ "What did you leave [before reading t]?"

Analisi:

- Movimento: $t \rightarrow \text{Spec-CP}$
- "before reading t" = adjunct $\rightarrow$ non L-marked $\rightarrow$ barriera
- PP adjunct = barriera addizionale
- Totale: 2 barriere $\rightarrow$ violazione Subjacency

7.14.2 Complex NP Constraint (CNPC)

*Teorema 7.14.2 (CNPC - formale):**

$\forall \alpha, \text{NP}$:

α è in complemento di N in NP $\rightarrow$

α non può essere estratto da NP

Formalizzazione via barriere:

[DP D [NP N [CP che ... t ...]]]

↑ _____ |

NP = blocking category (non L-marked)

$\rightarrow$ NP = barrier

$\rightarrow$ movimento viola Subjacency

Esempio 7.14.2:

✗ "Who did you hear [the rumor [that John saw t]]?"

✓ "Who did you hear [t saw John]?" (no complex NP)

7.14.3 Wh-islands

*Teorema 7.14.3 (Wh-island constraint):**

$\forall$ wh-phrase α in embedded interrogativa:

α non può essere ulteriormente estratto

Esempio 7.14.3:

✗ "What do you wonder [who bought t]?"

✓ "What do you think [t John bought t]?"

Formalizzazione:

[CP wh₁ C [IP ... wh₂ ...]]

↑ _____ |

CP con Spec occupato → barrier addizionale

Movimento di wh₂ attraversa due barriere → Subjacency violation

7.15 Reconstruction e Interpretazione

7.15.1 Fenomeni di Reconstruction

*Definizione 7.15.1 (Reconstruction):** Interpretazione di una frase moved come se fosse nella sua posizione originale (trace).

*Teorema 7.15.1 (Ambiti di reconstruction):**

∀ movimento M:

Interpretazione(moved-phrase) può fare riferimento a:

1. Posizione superficie (scope alto)
2. Posizione traccia (scope basso - reconstructed)

Esempio 7.15.1 (Scope reconstruction):

"How many books did everyone read?"

Lettura 1 (surface scope): $\exists > \forall$

"Qual è il numero n tale che tutti hanno letto n libri?"

Lettura 2 (reconstructed): $\forall > \exists$

"Per ogni persona, quanti libri ha letto?"

Struttura:

[CP [How many books]_i did [everyone read t_i]]

↓ reconstruction

[everyone read t_i]

7.15.2 Binding Reconstruction

*Teorema 7.15.2 (Binding reconstruction):**

Anafore/pronomi in frase moved mostrano effetti di binding

rispetto alla posizione del trace.

Esempio 7.15.2:

"[Which picture of himself_i]_j did John_i like t_j?"

Reconstruction:

John_i liked [which picture of himself_i]

↑ c-comando preservato via reconstruction

7.15.3 Teorema di Chomsky sulla Ricostruzione

*Teorema 7.15.3 (LF lowering):**

∀ movimento A'-movement at S-structure:

LF può "lower" la frase moved alla posizione del trace

per scopi interpretativi

Formalizzazione:

S-structure: [CP Phrase_i ... [IP ... t_i ...]]



LF representation: [CP t'_i ... [IP ... Phrase_i ...]]

7.16 Derivation by Phase: Sistema Completo

7.16.1 Definizione Formale di Fase

*Definizione 7.16.1 (Fase - estesa):**

$\text{Phase}(X) \Leftrightarrow X \in \{\text{CP}, v^*P, (\text{DP})\}$

Proprietà delle fasi:

$\forall$ Phase P:

1. Complete-interpretation(P) = TRUE

(interpretabile alle interfacce)

2. Propositional-or-event(P) = TRUE

(unità semantica)

3. Cyclic-Spell-Out(P) = TRUE

(invia complemento a PF/LF)

7.16.2 Algoritmo di Derivazione per Fase

Algoritmo 7.16.1 (Phase-based derivation):

Input: Numeration N (bag of lexical items)

Output: <PF, LF> pair o crash

Procedure:

1. Workspace $W \leftarrow \emptyset$
2. While $N \neq \emptyset$:
 - a. Select item from N
 - b. Merge into W
 - c. If Phase-complete(W):
 - Apply Spell-Out to Complement(Phase-head)
 - Transfer to PF/LF
 - Continue derivation with Phase-edge only
3. If no uninterpretable features remain:

Return <PF, LF>

Else: Crash

7.16.3 Multiple Spell-Out

*Teorema 7.16.1 (Multiple Spell-Out hypothesis):**

Transfer alle interfacce avviene ciclicamente:

[CP₂ ... [CP₁ ... [vP ... [VP ...]]]]

↑ ↑ ↑

SO₃ SO₂ SO₁

dove SO = Spell-Out operation

Vantaggi teorici:

1. Riduce carico computazionale (elimina material processed)
2. Spiega opacità (material transferred è inaccessibile)
3. Deriva PIC (Phase Impenetrability Condition)

7.16.4 Phase Sliding (Bobaljik & Wurmbrand)

Ipotesi 7.16.1 (Phase sliding): In contesti restructuring, le fasi possono "collassare":

"John wants [to leave]"

Non-restructuring: [vP want [CP to [vP leave]]]

↑ phase ↑ phase

Restructuring: [vP want to leave]

↑ single extended phase

Conseguenze:

1. Long-distance clitic climbing
2. Single event interpretation
3. Transparent per A-movement

7.17 Teoria dell'Informazione Linguistica

7.17.1 Structure vs. Order

*Principio 7.17.1 (Primacy of structure):**

Sintassi = Hierarchical structure

Ordine lineare = Secondary (imposto da PF)

*Teorema 7.17.1 (Independence of linear order):**

∀ struttura sintattica S:

Valid(S) è indipendente da linearizzazione(S)

Esempio 7.17.1:

Inglese: [VP read the book]

Giapponese: [VP the book read]

Stessa struttura gerarchica:

VP

/ \

V DP

7.17.2 Antisymmetry (Kayne)

*Ipotesi 7.17.1 (Linear Correspondence Axiom - LCA):**

$\forall X, Y: X \text{ precede } Y \text{ in ordine lineare} \Leftrightarrow X \text{ c-comanda } Y \text{ asimmetricamente}$

*Teorema 7.17.2 (Universalità SVO-like structure):**

Se LCA è corretta:

$\forall \text{ lingua } L: \text{underlying-order}(L) = \text{Spec-Head-Complement}$

Variazione apparente deriva da movimento:

SOV (giapponese): [VP O V] $\leftarrow$ movimento di O

VSO (irlandese): [VP V S O] $\leftarrow$ movimento di V

7.17.3 Teorema sulla Complessità Sintattica

*Definizione 7.17.1 (Complessità sintattica):**

$\text{Complexity}(S) = f(\text{depth}(S), \text{width}(S), \text{movement-steps}(S))$

dove:

- depth = massima profondità dell'albero
- width = numero massimo di figli per nodo
- movement-steps = numero di copie/tracce

*Teorema 7.17.3 (Limitazioni sulla complessità):**

$\forall$ lingua naturale L:

$\exists$ bound B: $\forall$ frase $s \in L$: $\text{Complexity}(s) \leq B \cdot |s|^k$

dove $k < 2$ (subquadratico)

Dimostrazione (Schema): Binary branching $\rightarrow$ depth = $O(\log n)$ Movement limitato da phases $\rightarrow$ steps = $O(n)$ Quindi Complexity = $O(n \log n)$ ■

7.18 Interfacce: PF e LF

7.18.1 Architettura a Y (Y-model)

Modello 7.18.1 (Y-model architecture):

Numeration

↓

Narrow Syntax

(CHL - Merge, Agree, Move)

↓

Spell-Out

/ \

PF LF

(sound) (meaning)

*Principio 7.18.1 (Bare output conditions):**

$\forall$ rappresentazione R alle interfacce:

$\text{legible}(R, \text{Interface}) = \text{TRUE}$

7.18.2 PF Interface: Linearizzazione e Fonologia

Operazioni PF:

1. Linearization: $\text{Tree} \rightarrow \text{String}$
2. Vocabulary Insertion: $\text{Abstract features} \rightarrow \text{Phonemes}$
3. Phonological rules: $/\text{underlying}/ \rightarrow [\text{surface}]$

*Teorema 7.18.1 (Linearizzazione totale):**

$\forall$ struttura sintattica S:

PF mappa S a sequenza totalmente ordinata

7.18.3 LF Interface: Interpretazione Semantica

Operazioni LF:

1. QR (Quantifier Raising): scope disambiguation
2. Type-shifting: category adjustment
3. Compositional interpretation: $[[S]] = f([[NP]], [[VP]])$

*Teorema 7.18.2 (Composizionalità stretta):**

$\forall$ nodo N con figli C_1, C_2 :

$[[N]] = f([[C_1]], [[C_2]])$

dove f è determinato da head di N

7.19 Variazione Cross-linguistica: Teoria Avanzata

7.19.1 Principles & Parameters Model

Framework:

UG = Principles $\cup$ Parameters

Principles = Universal constraints (invariant)

Parameters = Binary choices (variant)

Grammar_L = Principles + {Setting of Parameters for L}

Esempio 7.19.1 (Inventory of parameters):

1. Head-direction: {initial, final}
2. Pro-drop: {yes, no}
3. Wh-movement: {overt, covert}
4. V-to-I movement: {yes, no}
5. Null-topic: {yes, no}
- ...

Totale stimato: $\approx$ 50-100 parameters

7.19.2 Macro- vs. Micro-parameters

*Definizione 7.19.1 (Macro-parameter):** Parameter con effetti multipli e correlati:

Example: Pro-drop parameter

Effects:

- Null subject
- Free inversion
- Long wh-movement

- Rich agreement

*Definizione 7.19.2 (Micro-parameter):** Parameter con effetto limitato:

Example: Position of clitic "ne" in Italian dialects

Toscano: lo = ne leggo

Milanese: el = ne lesi

*Teorema 7.19.1 (Diminishing returns):**

Numero di lingue distinte $< 2^n$

(molti settings incompatibili o indistinguibili)

7.19.3 Clustering e Implicazioni

*Teorema 7.19.2 (Implicational universals):**

$\forall L_1, L_2$ lingue:

$\{\text{Parameter-settings}(L_1)\} \cap \{\text{Parameter-settings}(L_2)\} \neq \emptyset$

$\rightarrow \text{similar-properties}(L_1, L_2) > \text{baseline}$

Esempio empirico:

OV languages tend to have:

- Postpositions (not prepositions)
- Genitive-Noun (not Noun-Genitive)
- Adjective-Noun
- SOV order

Correlazione > 0.7 (Greenberg, Dryer)

Parte VI: Semantica Formale - Richard Montague

Capitolo 8: La Grammatica di Montague e la Semantica Intensionale

8.1 Fondamenti Filosofici della Semantica Formale

Richard Montague, nel suo programma di ricerca sviluppato principalmente negli anni Sessanta e Settanta, propose un approccio radicalmente diverso allo studio del linguaggio naturale, basato sull'ipotesi che non vi sia differenza teorica fondamentale tra le lingue naturali e i linguaggi formali della logica. Il suo celebre aforisma "Non vedo alcuna differenza teorica importante tra le lingue naturali e i linguaggi artificiali dei logici" riassume questa posizione metodologica.

Postulato 8.1.1 (Principio di composizionalità di Frege-Montague): Il significato di un'espressione complessa è una funzione dei significati dei suoi costituenti e del modo in cui sono combinati sintatticamente.

Formalmente, data una grammatica G che genera espressioni sintattiche, esiste una funzione di interpretazione $[[\cdot]]$ tale che:

$\forall \alpha \in \text{Exp}(G)$, se $\alpha = F(\beta_1, \beta_2, \dots, \beta_n)$ sintatticamente, allora $[[\alpha]] = F_{\text{sem}}([[\beta_1]], [[\beta_2]], \dots, [[\beta_n]])$

dove F è una regola sintattica e F_{sem} è la corrispondente regola semantica.

Definizione 8.1.1 (Modello per il linguaggio naturale): Un modello M per un linguaggio naturale è una struttura:

$M = \langle D, I, W, <, @ \rangle$

dove: (i) D è un dominio non vuoto di entità (l'universo del discorso) (ii) I è un insieme di indici (possibili mondi e tempi): $I = W \times T$ dove W è l'insieme dei mondi possibili e T è l'insieme dei tempi (iii) $<$ è una relazione di accessibilità su $W \times W$ che determina quali mondi sono accessibili da quali altri (iv) $@ \in W$ è il mondo attuale (v) Per ogni mondo $w \in W$ e tempo $t \in T$, abbiamo un'interpretazione delle costanti

Definizione 8.1.2 (Tipi semantici): I tipi semantici sono definiti ricorsivamente a partire dai tipi base e e t :

(i) e è il tipo delle entità (elementi di D) (ii) t è il tipo dei valori di verità ($\{0, 1\}$ o $\{\text{Vero}, \text{Falso}\}$) (iii) Se σ e τ sono tipi, allora $\langle \sigma, \tau \rangle$ è il tipo delle funzioni da oggetti di tipo σ a oggetti di tipo τ (iv) Se σ è un tipo, allora $\langle s, \sigma \rangle$ è il tipo delle intensioni di tipo σ (funzioni da indici a oggetti di tipo σ)

Formalmente, definiamo ricorsivamente l'insieme Types:

$\text{Types} ::= e \mid t \mid \langle \sigma, \tau \rangle \text{ per } \sigma, \tau \in \text{Types}$

Definizione 8.1.3 (Denotazioni per tipo): Per ogni tipo σ , definiamo l'insieme D_σ delle possibili denotazioni:

(i) $D_e = D$ (il dominio delle entità) (ii) $D_t = \{0, 1\}$ (iii) $D_{\{\langle\sigma, \tau\rangle\}} = [D_\sigma \rightarrow D_\tau]$ (l'insieme di tutte le funzioni da D_σ a D_τ) (iv) $D_{\{\langle s, \sigma \rangle\}} = [I \rightarrow D_\sigma]$ (funzioni da indici a denotazioni di tipo σ)

Esempio 8.1.1: Tipi semantici di diverse categorie sintattiche:

- Nome proprio (es. "Giovanni"): tipo e (entità)
- Verbo intransitivo (es. "dorme"): tipo $\langle e, t \rangle$ (funzione da entità a valori di verità)
- Nome comune (es. "uomo"): tipo $\langle e, t \rangle$
- Verbo transitivo (es. "ama"): tipo $\langle e, \langle e, t \rangle \rangle$ (funzione da entità a funzioni da entità a valori di verità)
- Determinante (es. "ogni"): tipo $\langle \langle e, t \rangle, \langle \langle e, t \rangle, t \rangle \rangle$
- Proposizione (es. "Giovanni dorme"): tipo t

8.2 Logica Intensionale del Tipo Superiore

Montague sviluppò una logica intensionale (Intensional Logic, IL) che estende la logica dei tipi superiori di Church con operatori modali e intensionali.

Definizione 8.2.1 (Linguaggio della Logica Intensionale): Il linguaggio IL è definito da:

Simboli primitivi:

- Variabili: per ogni tipo σ , un'infinità numerabile di variabili $x_\sigma, y_\sigma, z_\sigma, \dots$
- Costanti logiche: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists, \lambda, \square, \Diamond, \wedge, \vee$
- Operatori intensionali: $\wedge$ (intensione), $\vee$ (estensione)
- Costanti non logiche: per ogni tipo σ , un insieme (possibilmente vuoto) di costanti c_σ

Formule ben formate: Per ogni tipo σ , l'insieme ME_σ delle espressioni significative di tipo σ è definito induttivamente:

- (i) Se α è una variabile o costante di tipo σ , allora $\alpha \in ME_\sigma$
- (ii) Se $\phi, \psi \in ME_t$, allora $\neg\phi, (\phi \wedge \psi), (\phi \vee \psi), (\phi \rightarrow \psi), (\phi \leftrightarrow \psi) \in ME_t$
- (iii) Se $\phi \in ME_t$ e x è una variabile di tipo σ , allora $\forall x \phi, \exists x \phi \in ME_t$
- (iv) Se $\alpha \in ME_\tau$ e x è una variabile di tipo σ , allora $\lambda x[\alpha] \in ME_{\{\langle\sigma, \tau\rangle\}}$
- (v) Se $\alpha \in ME_{\{\langle\sigma, \tau\rangle\}}$ e $\beta \in ME_\sigma$, allora $\alpha(\beta) \in ME_\tau$ (applicazione funzionale)
- (vi) Se $\alpha \in ME_\sigma$, allora $\wedge\alpha \in ME_{\{\langle s, \sigma \rangle\}}$ (intensione di α)
- (vii) Se $\alpha \in ME_{\{\langle s, \sigma \rangle\}}$, allora $\vee\alpha \in ME_\sigma$ (estensione di α all'indice corrente)
- (viii) Se $\phi \in ME_t$, allora $\square\phi, \Diamond\phi \in ME_t$ (necessità e possibilità)

Definizione 8.2.2 (Semantica della Logica Intensionale): Data un'interpretazione I che assegna ad ogni costante non logica c_σ una intensione (elemento di $D_{\langle s, \sigma \rangle}$), e un assignment g che assegna ad ogni variabile x_σ un elemento di D_σ , la denotazione $\llbracket \alpha \rrbracket^{M, i, g}$ di un'espressione α all'indice i rispetto a M e g è definita induttivamente:

(i) $\llbracket c_\sigma \rrbracket^{M, i, g} = I(c_\sigma)(i)$ per costanti

(ii) $\llbracket x_\sigma \rrbracket^{M, i, g} = g(x_\sigma)$ per variabili

(iii) $\llbracket \neg \varphi \rrbracket^{M, i, g} = 1$ sse $\llbracket \varphi \rrbracket^{M, i, g} = 0$

(iv) $\llbracket \varphi \wedge \psi \rrbracket^{M, i, g} = 1$ sse $\llbracket \varphi \rrbracket^{M, i, g} = 1$ e $\llbracket \psi \rrbracket^{M, i, g} = 1$

(v) $\llbracket \forall x_\sigma \varphi \rrbracket^{M, i, g} = 1$ sse $\llbracket \varphi \rrbracket^{M, i, g[x \rightarrow d]} = 1$ per ogni $d \in D_\sigma$

(vi) $\llbracket \lambda x_\sigma [\alpha] \rrbracket^{M, i, g} =$ quella funzione $f \in D_{\langle \sigma, \tau \rangle}$ tale che per ogni $d \in D_\sigma$: $f(d) = \llbracket \alpha \rrbracket^{M, i, g[x \rightarrow d]}$

(vii) $\llbracket \alpha(\beta) \rrbracket^{M, i, g} = \llbracket \alpha \rrbracket^{M, i, g}(\llbracket \beta \rrbracket^{M, i, g})$

(viii) $\llbracket \wedge \alpha \rrbracket^{M, i, g} =$ quella funzione $f \in D_{\langle s, \sigma \rangle}$ tale che per ogni indice $j \in I$: $f(j) = \llbracket \alpha \rrbracket^{M, j, g}$

(ix) $\llbracket \vee \alpha \rrbracket^{M, i, g} = \llbracket \alpha \rrbracket^{M, i, g}(i)$

(x) $\llbracket \Box \varphi \rrbracket^{M, i, g} = 1$ sse per ogni $j \in I$ tale che $i < j$: $\llbracket \varphi \rrbracket^{M, j, g} = 1$

Teorema 8.2.1 (Equivalenza intensione-estensione): Per ogni α di tipo σ :

$$\llbracket \vee \alpha \rrbracket^{M, i, g} = \llbracket \alpha \rrbracket^{M, i, g}$$

Dimostrazione: $\llbracket \vee \alpha \rrbracket^{M, i, g} = \llbracket \wedge \alpha \rrbracket^{M, i, g}(i)$ (per la clausola (ix)) = (quella funzione f tale che $f(j) = \llbracket \alpha \rrbracket^{M, j, g}$)(i) (per la clausola (viii)) = $\llbracket \alpha \rrbracket^{M, i, g}$ ■

Esempio 8.2.1 (Distinzione intensione-estensione):

Consideriamo le frasi: (a) "La capitale dell'Italia è Roma" (b) "Necessariamente, la capitale dell'Italia è Roma"

Sia $i_0 = \langle w_0, t_0 \rangle$ l'indice attuale. Abbiamo:

$$\llbracket \text{la capitale dell'Italia} \rrbracket^{M, i_0, g} = \text{quella città che è capitale dell'Italia in } w_0 \text{ al tempo } t_0 = \text{Roma}$$

Ma:

$$\llbracket \wedge (\text{la capitale dell'Italia}) \rrbracket^{M, i_0, g} = \text{quella funzione } f \text{ tale che } f(\langle w, t \rangle) = \text{la capitale dell'Italia in } w \text{ al tempo } t$$

In un mondo possibile w_1 diverso, potrebbe essere che la capitale dell'Italia sia Milano. Quindi:

$\llbracket \Box [\text{la capitale dell'Italia} = \text{Roma}] \rrbracket^{\{M, i_0, g\}} = 0$

perché esistono mondi accessibili dove la capitale dell'Italia non è Roma.

8.3 PTQ: "The Proper Treatment of Quantification in Ordinary English"

Il lavoro seminale di Montague "The Proper Treatment of Quantification in Ordinary English" (PTQ, millenovecentosettantatré) fornisce un'analisi compositiva completa di un frammento significativo dell'inglese, inclusi quantificatori, atteggiamenti proposizionali, e tempi verbali.

Definizione 8.3.1 (Sintassi di PTQ): Il frammento PTQ è definito attraverso un insieme di categorie sintattiche e regole di formazione.

Categorie sintattiche:

- t: termini (term phrases, includendo nomi propri e descrizioni definite)
- IV: verbi intransitivi (intransitive verb phrases)
- TV: verbi transitivi (transitive verb phrases)
- CN: nomi comuni (common noun phrases)
- S: frasi (sentences)

Regole sintattiche (selezione):

S_1 : Se $\alpha \in Pt$ e $\delta \in PIV$, allora $F_1(\alpha, \delta) \in PS$, dove $F_1(\alpha, \delta) = \alpha \delta$ ("Giovanni dorme")

S_2 : Se $\phi, \psi \in PS$, allora $F_2(\phi, \psi) \in PS$, dove $F_2(\phi, \psi) = \phi$ e ψ (congiunzione di frasi)

S_3 : Se $\alpha \in Pt$, allora $F_3(\alpha) \in Pt$, dove $F_3(\alpha) = \text{ogni } \alpha$ (quantificatore universale)

S_4 : Se $\alpha \in PCN$ e $\delta \in PIV$, allora $F_4(\alpha, \delta) \in Pt$, dove $F_4(\alpha, \delta) = \text{un/una } \alpha \text{ tale che } \delta$ (descrizioni indefinite)

... (in totale, PTQ ha diciassette regole sintattiche)

Definizione 8.3.2 (Traduzione in IL): Ad ogni regola sintattica F_n corrisponde una regola di traduzione T_n che mappa espressioni sintattiche in formule della Logica Intensionale.

Traduzione dei termini (regola S_3):

Se $\alpha \in Pt$ con traduzione α' , allora $T_3(\alpha') = \lambda P[\forall x[\alpha'(\lambda y[y = x]) \rightarrow P(\lambda x)]]$

dove P è una variabile di tipo $\langle\langle s, e \rangle, t \rangle$ (proprietà di entità individuali).

Esempio 8.3.1: Traduzione di "Ogni uomo dorme"

Sintassi:

- "uomo" $\in PCN$
- "dorme" $\in PIV$
- "ogni uomo" $\in Pt$ per S_3
- "Ogni uomo dorme" $\in PS$ per S_1

Traduzione:

- $[[\text{uomo}]]' = \text{uomo}$ (costante di tipo $\langle s, \langle e, t \rangle \rangle$)
- $[[\text{dorme}]]' = \text{^dorme}$ (tipo $\langle s, \langle e, t \rangle \rangle$)

Per "ogni uomo": $T_3(\text{uomo}) = \lambda P[\forall x[\text{uomo}(\lambda y[y = x]) \rightarrow P(^x)] = \lambda P[\forall x[\text{uomo}'(x) \rightarrow P(^x)]]$

dove $\text{uomo}'(x)$ abbrevia $\forall y(\text{uomo}(^y x))$.

Per "Ogni uomo dorme": $T_1(\lambda P[\forall x[\text{uomo}'(x) \rightarrow P(^x)]], \text{^dorme}) = \lambda P[\forall x[\text{uomo}'(x) \rightarrow P(^x)]] = \lambda P[\forall x[\text{uomo}'(x) \rightarrow \forall z[\text{dorme}'(z)]] = \forall x[\text{uomo}'(x) \rightarrow \text{dorme}'(x)]$

che si legge: "Per ogni x, se x è un uomo, allora x dorme".

Teorema 8.3.1 (Composizionalità di PTQ): Per ogni espressione α generata dalle regole sintattiche di PTQ, la sua traduzione $[[\alpha]]'$ in IL è determinata univocamente e composizionalmente dalla sua struttura sintattica.

Dimostrazione: Per induzione strutturale sulla derivazione di α . Ogni regola sintattica F_n ha una corrispondente regola di traduzione T_n , e le traduzioni dei costituenti determinano univocamente la traduzione del tutto. ■

8.4 Trattamento dei Quantificatori

Una delle innovazioni più significative di Montague è il trattamento uniforme dei quantificatori come espressioni di tipo $\langle \langle e, t \rangle, t \rangle$, ovvero come insiemi di insiemi (o proprietà di proprietà).

Definizione 8.4.1 (Quantificatori generalizzati): Un quantificatore generalizzato Q di tipo $\langle 1 \rangle$ è un elemento di $D_{\{\langle \langle e, t \rangle, t \rangle\}}$, ovvero un insieme di insiemi di entità.

Per una interpretazione I e indice i :

$$[[Q]]^{M, i, g} \in \mathcal{P}(\mathcal{P}(D))$$

Esempio 8.4.1: Interpretazioni di quantificatori determiners:

(i) $[[\text{ogni}]](A)(B) = 1$ sse $A \subseteq B$ Formalmente: $[[\text{ogni}]] = \lambda A \lambda B[A \subseteq B]$

(ii) $[[\text{qualche}]](A)(B) = 1$ sse $A \cap B \neq \emptyset$ Formalmente: $[[\text{qualche}]] = \lambda A \lambda B[A \cap B \neq \emptyset]$

(iii) $[[\text{nessun}]](A)(B) = 1$ sse $A \cap B = \emptyset$ Formalmente: $[[\text{nessun}]] = \lambda A \lambda B[A \cap B = \emptyset]$

(iv) $[[\text{la maggior parte dei}]](A)(B) = 1$ sse $|A \cap B| > |A \setminus B|$ Formalmente: $[[\text{la maggior parte dei}]] = \lambda A \lambda B[|A \cap B| > |A \setminus B|]$

Definizione 8.4.2 (Quantificatori conservativi): Un quantificatore Q è conservativo se e solo se:

$$Q(A)(B) = Q(A)(A \cap B)$$

per ogni $A, B \subseteq D$.

Teorema 8.4.1 (Universalità della conservatività): Tutti i determinanti nelle lingue naturali denotano quantificatori conservativi.

Dimostrazione (Evidenza empirica): Verifichiamo per i quantificatori comuni:

(i) "Ogni": $\text{Ogni}(A)(B) = 1$ sse $A \subseteq B$ sse $A \subseteq A \cap B$ sse $A = A \cap B$ sse $\text{Ogni}(A)(A \cap B) = 1$

(ii) "Qualche": $\text{Qualche}(A)(B) = 1$ sse $A \cap B \neq \emptyset$ sse $A \cap (A \cap B) \neq \emptyset$ sse $\text{Qualche}(A)(A \cap B) = 1$

La conservatività cattura il fatto che frasi come "Ogni uomo dorme" parlano solo degli uomini, non di entità che non sono uomini. ■

Definizione 8.4.3 (Monotonia): Un quantificatore Q è:

(i) Monotono crescente a destra se: $Q(A)(B)$ e $B \subseteq B'$ implica $Q(A)(B')$ (ii) Monotono decrescente a destra se: $Q(A)(B)$ e $B' \subseteq B$ implica $Q(A)(B')$ (iii) Monotono crescente a sinistra se: $Q(A)(B)$ e $A \subseteq A'$ implica $Q(A')(B)$ (iv) Monotono decrescente a sinistra se: $Q(A)(B)$ e $A' \subseteq A$ implica $Q(A')(B)$

Esempio 8.4.2: Classificazione per monotonia:

- "Ogni" è $\downarrow$ -mono a sinistra, $\uparrow$ -mono a destra
- "Qualche" è $\uparrow$ -mono a sinistra, $\downarrow$ -mono a destra
- "Nessun" è $\downarrow$ -mono a sinistra, $\downarrow$ -mono a destra
- "Esattamente tre" è non-mono in entrambe le direzioni

8.5 Atteggiamenti Proposizionali e de re/de dicto

Montague fornisce un trattamento formale della distinzione tra interpretazioni de re e de dicto negli atteggiamenti proposizionali.

Definizione 8.5.1 (Verbi di atteggiamento proposizionale): Un verbo come "credere", "sapere", "cercare" prende come argomento una proposizione (tipo $\langle s, t \rangle$) e restituisce una proprietà di individui (tipo $\langle e, t \rangle$).

$[[\text{credere}]] : D_e \rightarrow D_{\{\langle s, t \rangle, t\}}$

Formalmente: $[[\text{credere}]]^{\{M, i, g\}} = \lambda x \lambda p [\text{Cred}_x(i) \subseteq \{j \in I : p(j) = 1\}]$

dove $\text{Cred}_x(i)$ è l'insieme dei mondi compatibili con ciò che x crede all'indice i .

Esempio 8.5.1: Ambiguità de re/de dicto:

"Giovanni cerca un unicorno"

Interpretazione de dicto: Giovanni è nella relazione di ricerca con la proprietà di essere un unicorno (anche se gli unicorni non esistono).

$\exists x [\text{unicorno}'(x) \wedge \text{cerca}'(\text{Giovanni}', \lambda y [y = x])]$

Interpretazione de re: C'è un particolare unicorno che Giovanni sta cercando.

cerca'(Giovanni', $\lambda y[\exists x[\text{unicorno}'(x) \wedge y = x]]$)

Definizione 8.5.2 (Innalzamento di tipo): Per gestire l'ambiguità, Montague introduce regole di innalzamento di tipo (type raising) che convertono termini di tipo e in espressioni di tipo $\langle e, t \rangle$:

$\text{raise}(\alpha) = \lambda P[P(\alpha)]$

dove α è di tipo e e P è una variabile di tipo $\langle e, t \rangle$.

Teorema 8.5.1 (Equivalenza logica delle letture): Le due letture de re e de dicto di "Giovanni cerca un unicorno" sono logicamente indipendenti:

de dicto $\neq$ de re e de re $\neq$ de dicto

Dimostrazione: Costruiamo modelli per mostrare l'indipendenza.

Modello 1 (de dicto vero, de re falso): Sia $D = \{g, u_1, u_2, \dots\}$ dove g = Giovanni e gli u_i sono pseudo-unicorni (che Giovanni crede essere unicorni ma non lo sono). In ogni mondo nel mondo della credenza di Giovanni, esiste un unicorno, ma nel mondo attuale non esistono unicorni. Quindi de dicto è vero ma de re è falso.

Modello 2 (de re vero, de dicto falso): Giovanni sta cercando una particolare creatura che è in realtà un unicorno (all'insaputa di Giovanni). Quindi de re è vero, ma de dicto potrebbe essere falso se Giovanni non crede di stare cercando un unicorno. ■

8.6 Tempi Verbali e Aspetto

Montague tratta i tempi verbali come operatori che spostano l'indice temporale di valutazione.

Definizione 8.6.1 (Operatori temporali): Introduciamo operatori temporali nella logica intensionale:

- $P \phi$ (Past): " ϕ era vero"
- $F \phi$ (Future): " ϕ sarà vero"
- $H \phi$ (Past perfect): " ϕ è sempre stato vero"
- $G \phi$ (Future perfect): " ϕ sarà sempre vero"

Semantica:

$\llbracket P \phi \rrbracket^{M, \langle w, t \rangle, g} = 1$ sse $\exists t' < t : \llbracket \phi \rrbracket^{M, \langle w, t' \rangle, g} = 1$

$\llbracket F \phi \rrbracket^{M, \langle w, t \rangle, g} = 1$ sse $\exists t' > t : \llbracket \phi \rrbracket^{M, \langle w, t' \rangle, g} = 1$

$\llbracket H \phi \rrbracket^{M, \langle w, t \rangle, g} = 1$ sse $\forall t' < t : \llbracket \phi \rrbracket^{M, \langle w, t' \rangle, g} = 1$

$\llbracket G \phi \rrbracket^{M, \langle w, t \rangle, g} = 1$ sse $\forall t' > t : \llbracket \phi \rrbracket^{M, \langle w, t' \rangle, g} = 1$

Esempio 8.6.1: "Giovanni ha dormito"

$[[\text{ha dormito}]] = \wedge P[\text{dorme}]$

dove l'operatore P sposta la valutazione a un tempo passato.

Definizione 8.6.2 (Aspetto perfettivo vs imperfettivo):

L'aspetto perfettivo presenta un evento come completo e limitato:

$\text{PERF}(\varphi)$ indica che l'evento φ è avvenuto completamente in un intervallo temporale definito.

L'aspetto imperfettivo presenta un evento come in corso:

$\text{IMPERF}(\varphi)$ indica che l'evento φ era in corso in un intervallo temporale.

Teorema 8.6.1 (Paradosso imperfettivo): L'aspetto imperfettivo non implica l'aspetto perfettivo corrispondente:

$\text{IMPERF}(\varphi) \not\models \text{PERF}(\varphi)$

Dimostrazione (Controesempio): "Giovanni stava attraversando la strada" (imperfettivo) non implica "Giovanni ha attraversato la strada" (perfettivo), perché Giovanni potrebbe essere stato interrotto a metà strada. ■

Capitolo 8 Espanso: Semantica di Montague - Fondamenti Avanzati

Parte I: Logica Intensionale del Tipo Superiore - Trattamento Completo

8.1 Sistema di Tipi: Teoria Formale Completa

8.1.1 Algebra dei Tipi

*Definizione 8.1.1 (Grammatica dei tipi - BNF estesa):**

$\tau ::= e$	(entità)
t	(valori di verità)
s	(indici/mondi possibili)
$\langle \tau, \tau \rangle$	(tipi funzionali)
$\langle s, \tau \rangle$	(intensioni)

$| \tau \times \tau$ (prodotto cartesiano)

$| \tau + \tau$ (somma disgiunta - tipo unione)

Teorema 8.1.1 (Equazioni tra tipi):*

1. $\langle \sigma, \langle \tau, \rho \rangle \rangle \equiv \langle \sigma \times \tau, \rho \rangle$ (currying)
2. $\langle \sigma + \tau, \rho \rangle \equiv \langle \sigma, \rho \rangle \times \langle \tau, \rho \rangle$ (distributività)
3. $\langle s, \langle \sigma, \tau \rangle \rangle \equiv \langle \sigma, \langle s, \tau \rangle \rangle$ (commutatività)

ità intensioni)

Dimostrazione (caso 1 - currying): Definiamo isomorfismi:

$\text{curry}: [\sigma \times \tau \rightarrow \rho] \rightarrow [\sigma \rightarrow [\tau \rightarrow \rho]]$

$\text{curry}(f)(x)(y) = f(\langle x, y \rangle)$

$\text{uncurry}: [\sigma \rightarrow [\tau \rightarrow \rho]] \rightarrow [\sigma \times \tau \rightarrow \rho]$

$\text{uncurry}(g)(\langle x, y \rangle) = g(x)(y)$

Verifichiamo che sono inverse:

$\text{uncurry}(\text{curry}(f))(\langle x, y \rangle) = \text{curry}(f)(x)(y) = f(\langle x, y \rangle) = f$

$\text{curry}(\text{uncurry}(g))(x)(y) = \text{uncurry}(g)(\langle x, y \rangle) = g(x)(y) = g$ ■

8.1.2 Denotazioni e Domini Semantici

Costruzione 8.1.1 (Domini per ogni tipo):*

Per un modello $M = \langle D, I, W, <, @ \rangle$:

$D_e = D$ (dominio delle entità)

$D_t = \{0, 1\}$ (Booleani)

$D_s = W \times T$ (indici = mondi $\times$ tempi)

$$D_{\langle\sigma,\tau\rangle} = [D_\sigma \rightarrow D_\tau] \quad (\text{spazio funzionale})$$

$$D_{\langle s,\sigma\rangle} = [D_s \rightarrow D_\sigma] \quad (\text{intensioni})$$

$$D_{\sigma \times \tau} = D_\sigma \times D_\tau \quad (\text{prodotto cartesiano})$$

$$D_{\sigma + \tau} = (\{L\} \times D_\sigma) \cup (\{R\} \times D_\tau) \quad (\text{somma disgiunta taggata})$$

*Teorema 8.1.2 (Cardinalità dei domini):**

$$|D_e| = \kappa \quad (\text{cardinalità arbitraria})$$

$$|D_t| = 2$$

$$|D_{\langle\sigma,\tau\rangle}| = |D_\tau|^{|D_\sigma|}$$

$$|D_{\langle s,\sigma\rangle}| = |D_\sigma|^{|D_s|}$$

$$\text{Se } |W| = |T| = \aleph_0 \text{ e } \kappa = \aleph_0:$$

$$|D_{\langle s,\langle e,t \rangle \rangle}| = (2^{\aleph_0})^{\aleph_0} = 2^{\aleph_0 \cdot \aleph_0} = 2^{\aleph_0} = \aleph_c$$

8.1.3 Lambda Calcolo Tipato per IL

Sintassi estesa del Lambda Calcolo:

$$t ::= x_\tau \quad (\text{variabile di tipo } \tau)$$

$$| c_\tau \quad (\text{costante di tipo } \tau)$$

$$| (t \ t) \quad (\text{applicazione})$$

$$| (\lambda x_\sigma. t_\tau) \quad (\text{astrazione})$$

$$| \langle t, t \rangle \quad (\text{coppia})$$

$$| \pi_1(t) \mid \pi_2(t) \quad (\text{proiezioni})$$

$$| \text{inj}_1(t) \mid \text{inj}_2(t) \quad (\text{iniezioni in somma})$$

$$| \text{case } t \text{ of } \text{inj}_1(x) \rightarrow t \mid \text{inj}_2(y) \rightarrow t \quad (\text{pattern matching})$$

$$| ^t \quad (\text{intensione})$$

$$| \forall t \quad (\text{estensione})$$

Regole di tipo estese:

$$\Gamma \vdash t: \sigma \quad \Gamma \vdash s: \tau$$

 (pair)

$$\Gamma \vdash \langle t, s \rangle: \sigma \times \tau$$

$$\Gamma \vdash t: \sigma \times \tau \quad \Gamma \vdash t: \sigma \times \tau$$

 (π_1)

 (π_2)

$$\Gamma \vdash \pi_1(t): \sigma \quad \Gamma \vdash \pi_2(t): \tau$$

$$\Gamma \vdash t: \sigma$$

 (inj_i)

$$\Gamma \vdash \text{inj}_1(t): \sigma + \tau$$

$$\Gamma \vdash t: \sigma + \tau \quad \Gamma, x: \sigma \vdash s_1: \rho \quad \Gamma, y: \tau \vdash s_2: \rho$$

 (case)

$$\Gamma \vdash \text{case } t \text{ of } \text{inj}_1(x) \rightarrow s_1 \mid \text{inj}_2(y) \rightarrow s_2: \rho$$
8.2 Semantica Denotazionale Completa**8.2.1 Interpretazione dei Termini**

*Definizione 8.2.1 (Interpretazione $\llbracket \cdot \rrbracket$):**

$$\llbracket \cdot \rrbracket^{\{M, i, g\}}: \text{Terms}_\tau \rightarrow D_\tau$$

Per ogni termine tipato $t: \Gamma \vdash t: \tau$, definiamo:

Variabili e costanti:

$$\llbracket x_\tau \rrbracket^{\{M, i, g\}} = g(x) \quad (\text{assignment})$$

$$\llbracket c_\tau \rrbracket^{M,i,g} = I(c)(i) \quad (\text{interpretazione} + \text{indice})$$

Applicazione e astrazione:

$$\llbracket t s \rrbracket^{M,i,g} = \llbracket t \rrbracket^{M,i,g}(\llbracket s \rrbracket^{M,i,g})$$

$$\llbracket \lambda x_\sigma. t_\tau \rrbracket^{M,i,g} = \text{la funzione } f: D_\sigma \rightarrow D_\tau \text{ tale che}$$

$$\forall d \in D_\sigma: f(d) = \llbracket t \rrbracket^{M,i,g}[x \mapsto d]$$

Prodotti:

$$\llbracket \langle t, s \rangle \rrbracket^{M,i,g} = (\llbracket t \rrbracket^{M,i,g}, \llbracket s \rrbracket^{M,i,g})$$

$$\llbracket \pi_1(t) \rrbracket^{M,i,g} = \text{primo componente di } \llbracket t \rrbracket^{M,i,g}$$

$$\llbracket \pi_2(t) \rrbracket^{M,i,g} = \text{secondo componente di } \llbracket t \rrbracket^{M,i,g}$$

Somme:

$$\llbracket \text{inj}_1(t) \rrbracket^{M,i,g} = (L, \llbracket t \rrbracket^{M,i,g})$$

$$\llbracket \text{inj}_2(t) \rrbracket^{M,i,g} = (R, \llbracket t \rrbracket^{M,i,g})$$

$$\llbracket \text{case } t \text{ of } \text{inj}_1(x) \rightarrow s_1 \mid \text{inj}_2(y) \rightarrow s_2 \rrbracket^{M,i,g} =$$

$$\text{if } \llbracket t \rrbracket = (L, v): \llbracket s_1 \rrbracket^{M,i,g}[x \mapsto v]$$

$$\text{if } \llbracket t \rrbracket = (R, v): \llbracket s_2 \rrbracket^{M,i,g}[y \mapsto v]$$

Operatori intensionali:

$$\llbracket \lambda^t \rrbracket^{M,i,g} = \lambda j \in D_s. \llbracket t \rrbracket^{M,j,g}$$

$$\llbracket \forall t \rrbracket^{M,i,g} = \llbracket t \rrbracket^{M,i,g}(i)$$

8.2.2 Teoremi Fondamentali

*Teorema 8.2.1 (Sostituzione):**

$$\forall t, s, x: \llbracket t[s/x] \rrbracket^{M,i,g} = \llbracket t \rrbracket^{M,i,g[x \mapsto \llbracket s \rrbracket^{M,i,g}]]}$$

Dimostrazione: Per induzione sulla struttura di t . ■

*Teorema 8.2.2 (α -equivalenza):**

Se $t =_{\alpha} s$ (t e s sono α -equivalenti), allora $\llbracket t \rrbracket = \llbracket s \rrbracket$

*Teorema 8.2.3 (β -riduzione preserva significato):**

Se $t \rightarrow_{\beta} s$, allora $\llbracket t \rrbracket = \llbracket s \rrbracket$

Dimostrazione: Caso critico: $(\lambda x.t)s \rightarrow_{\beta} t[s/x]$

$$\begin{aligned} \llbracket (\lambda x.t)s \rrbracket^{M,i,g} &= \llbracket \lambda x.t \rrbracket^{M,i,g}(\llbracket s \rrbracket^{M,i,g}) \\ &= (\lambda d. \llbracket t \rrbracket^{M,i,g[x \mapsto d]}) (\llbracket s \rrbracket^{M,i,g}) \\ &= \llbracket t \rrbracket^{M,i,g[x \mapsto \llbracket s \rrbracket^{M,i,g}]]} \\ &= \llbracket t[s/x] \rrbracket^{M,i,g} \quad (\text{per Teorema 8.2.1}) \quad \blacksquare \end{aligned}$$

*Teorema 8.2.4 (Identità intensione-estensione):**

$$\llbracket \vee^t \rrbracket^{M,i,g} = \llbracket t \rrbracket^{M,i,g}$$

Dimostrazione:

$$\begin{aligned} \llbracket \vee^t \rrbracket^{M,i,g} &= \llbracket \vee^t \rrbracket^{M,i,g}(i) \\ &= (\lambda j. \llbracket t \rrbracket^{M,j,g})(i) \\ &= \llbracket t \rrbracket^{M,i,g} \quad \blacksquare \end{aligned}$$

8.3 Teoria delle Intensioni: Formalizzazione Completa

8.3.1 Intensioni come Funzioni

*Definizione 8.3.1 (Intensione vs. Estensione):**

Per ogni espressione α di tipo τ :

$$\text{Intensione}(\alpha) = \lambda\alpha: \langle s, \tau \rangle$$

= funzione da indici a estensioni

$$\text{Estensione}(\alpha) \text{ at } i = \bigvee (\lambda\alpha) \text{ at } i$$

= valore di α all'indice i

*Teorema 8.3.1 (Principio di composizionalità intensionale):**

$$\forall \alpha, \beta: \llbracket \alpha(\beta) \rrbracket^i = \llbracket \alpha \rrbracket^i(\llbracket \beta \rrbracket^i)$$

Ma:

$$\llbracket \lambda(\alpha(\beta)) \rrbracket \neq \llbracket \lambda\alpha \rrbracket(\llbracket \lambda\beta \rrbracket) \text{ in generale}$$

Serve composizione intensionale:

$$\llbracket \lambda(\alpha(\beta)) \rrbracket = \lambda i. \llbracket \alpha \rrbracket^i(\llbracket \beta \rrbracket^i)$$

8.3.2 Operatori Intensionali

*Definizione 8.3.2 (Modalità):**

$$\text{Necessità: } \Box\varphi \Leftrightarrow \forall i' \text{ accessibile da } i: \llbracket \varphi \rrbracket^{i'} = 1$$

$$\text{Possibilità: } \Diamond\varphi \Leftrightarrow \exists i' \text{ accessibile da } i: \llbracket \varphi \rrbracket^{i'} = 1$$

Semantica formale:

$$\llbracket \Box\varphi \rrbracket^{M,i,g} = 1 \Leftrightarrow \forall j \in W: i R j \Rightarrow \llbracket \varphi \rrbracket^{M,j,g} = 1$$

$$\llbracket \Diamond\varphi \rrbracket^{M,i,g} = 1 \Leftrightarrow \exists j \in W: i R j \wedge \llbracket \varphi \rrbracket^{M,j,g} = 1$$

dove $R \subseteq W \times W$ è la relazione di accessibilità.

Teorema 8.3.2 (Interdefinibilità modale):*

$$\llbracket \Box \varphi \rrbracket = \llbracket \neg \Diamond \neg \varphi \rrbracket$$

$$\llbracket \Diamond \varphi \rrbracket = \llbracket \neg \Box \neg \varphi \rrbracket$$

Teorema 8.3.3 (Assiomi modali in IL):*

Se R è:

- Riflessiva $\rightarrow$ T: $\Box \varphi \rightarrow \varphi$
- Transitiva $\rightarrow$ 4: $\Box \varphi \rightarrow \Box \Box \varphi$
- Simmetrica $\rightarrow$ B: $\varphi \rightarrow \Box \Diamond \varphi$
- Euclidea $\rightarrow$ 5: $\Diamond \varphi \rightarrow \Box \Diamond \varphi$

8.3.3 Atteggiamenti Proporzionali

Definizione 8.3.3 (Verbi di atteggiamento):*

$$\llbracket \text{credere} \rrbracket^{\{M,i,g\}}: D_e \rightarrow D_{\langle s,t \rangle} \rightarrow D_t$$

$$\llbracket \text{credere} \rrbracket(x)(p)(i) = 1 \Leftrightarrow$$

$$\forall j \in \text{Dox}_x(i): p(j) = 1$$

dove $\text{Dox}_x(i)$ = mondi compatibili con le credenze di x in i (doxastic alternatives).

Esempio 8.3.1 (Credenza e necessità soggettiva):

"John crede che φ " $\approx$ "In tutti i mondi compatibili con le credenze di John, φ "

Formale:

$$\llbracket \text{crede} \rrbracket(\text{john})(\varphi)(i) = 1 \Leftrightarrow$$

$$\forall j \in \text{Dox}_{\{\text{john}\}}(i): \llbracket \varphi \rrbracket^j = 1$$

Teorema 8.3.4 (Non-veridicità):*

"x crede φ " $\neq \varphi$

Cioè: $\llbracket \text{crede} \rrbracket(x)(\wedge \varphi) \neq \varphi$

8.4 Quantificazione: Teoria Avanzata

8.4.1 Quantificatori Generalizzati - Formalizzazione Completa

*Definizione 8.4.1 (Gerarchia dei tipi quantificazionali):**

Tipo $\langle 1 \rangle$:

$$Q: D_{\langle e, t \rangle} \rightarrow D_{\langle e, t \rangle} \rightarrow D_t$$

Interpretazione: $Q(A)(B) = \text{verità/falsità}$

Tipo $\langle 1, 1 \rangle$ (quantificatori binari):

$$Q: D_{\langle e, t \rangle} \rightarrow D_{\langle e, t \rangle} \rightarrow D_{\langle e, t \rangle} \rightarrow D_t$$

Esempio: "More As than Bs are C"

*Teorema 8.4.1 (Teorema di Barwise-Cooper - esteso):**

Tutti i determinanti del linguaggio naturale soddisfano:

Conservatività:

$$Q(A)(B) \Leftrightarrow Q(A)(A \cap B)$$

1.

Extension:

$Q(A)(B)$ dipende solo da $|A \cap B|$, $|A - B|$, $|B - A|$ (in lingue con determiner)

2.

Dimostrazione (Conservatività per "ogni"):

$$\llbracket \text{ogni} \rrbracket(A)(B) = 1 \Leftrightarrow A \subseteq B$$

$$\Leftrightarrow A \subseteq A \cap B \quad (\text{poiché } A \cap B \subseteq B)$$

$$\Leftrightarrow A = A \cap B$$

$$\Leftrightarrow \llbracket \text{ogni} \rrbracket(A)(A \cap B) = 1$$

I

8.4.2 Proprietà Logiche dei Quantificatori

*Definizione 8.4.2 (Monotonia):**

Monotonia crescente a destra ($\uparrow$ -MON-R):

$$Q(A)(B) \wedge B \subseteq B' \Rightarrow Q(A)(B')$$

Monotonia decrescente a destra ($\downarrow$ -MON-R):

$$Q(A)(B) \wedge B' \subseteq B \Rightarrow Q(A)(B')$$

Analogamente per sinistra ($\uparrow$ -MON-L, $\downarrow$ -MON-L).

*Teorema 8.4.2 (Classificazione dei quantificatori comuni):**

Quantificatore	MON-L	MON-R
ogni	$\downarrow$	$\uparrow$
qualche	$\uparrow$	$\uparrow$
nessun	$\downarrow$	$\downarrow$
la maggioranza	$\uparrow$	$\uparrow$
pochi	$\downarrow$	$\downarrow$
esattamente n	-	-

*Teorema 8.4.3 (Licensing di NPI):**

Negative Polarity Items (any, ever, ...) sono licenziati in contesti $\downarrow$ -MON

Formalmente:

$\text{NPI-licensed}(\varphi) \Leftrightarrow \exists \text{ operatore } O: \downarrow\text{-MON}(O) \wedge \text{in-scope}(O, \varphi)$

Esempio 8.4.1:

✓ "Nessuno ha mai visto un unicorno"

(nessuno = $\downarrow$ -MON $\rightarrow$ licenzia "mai")

✗ "*Qualcuno ha mai visto un unicorno"

(qualcuno = $\uparrow$ -MON $\rightarrow$ non licenzia "mai")

8.4.3 Scope e Ambiguità

*Definizione 8.4.3 (Quantifier Raising - QR):**

LF-movimento dei quantificatori per disambiguare scope

Esempio:

"Ogni studente ama qualche professore"

$\text{LF}_1: [\text{Ogni studente}]_i [\text{qualche professore}]_{\square} [t_i \text{ ama } t_{\square}]$

$\forall > \exists$: per ogni studente, c'è un (possibilmente diverso) professore

$\text{LF}_2: [\text{Qualche professore}]_{\square} [\text{ogni studente}]_i [t_i \text{ ama } t_{\square}]$

$\exists > \forall$: c'è uno specifico professore che ogni studente ama

*Teorema 8.4.4 (Scope ambiguity formula):**

Per n quantificatori, numero di letture di scope = $n!$

(nel caso peggiore, ignorando island constraints)

*Definizione 8.4.4 (Relazione di c-comando e scope):**

Default Scope Principle:

α ha scope su β se α c-comanda β alla LF

Formalmente:

$\text{scope}(\alpha) > \text{scope}(\beta) \Leftrightarrow \text{c-comanda-LF}(\alpha, \beta)$

8.5 Intentionality e Opacità Referenziale

8.5.1 Distinzione de dicto / de re

*Definizione 8.5.1 (Letture de dicto e de re):**

De dicto (about the sense):

John seeks [a unicorn]

$[[\text{cerca}]](\text{john})(\lambda x. \text{unicorno}(x))$

Interpretazione: John è nella relazione di ricerca con la proprietà di essere unicorno

De re (about the thing):

$[A \text{ unicorn}]_i$ (John seeks t_i)

$\exists x[\text{unicorno}(x) \wedge [[\text{cerca}]](\text{john})(\lambda y[y = x])]$

Interpretazione: Esiste un unicorno specifico che John cerca

*Teorema 8.5.1 (Indipendenza logica):**

de-dicto $\neq$ de-re

de-re $\neq$ de-dicto

Dimostrazione (de-dicto $\neq$ de-re): Modello M:

- John cerca un unicorno (concettualmente)
- Non esistono unicorni: $\forall x \neg \text{unicorno}(x)$

Allora:

- de-dicto: TRUE (John ha l'intenzione di cercare)
- de-re: FALSE ($\nexists x$ che John cerca)

Quindi de-dicto non implica de-re. ■

8.5.2 Contesti Opachi

*Definizione 8.5.2 (Contesto opaco):**

Un contesto $C[\]$ è opaco se:

$$\alpha = \beta \neq C[\alpha] = C[\beta]$$

Cioè: sostituzione di termini coreferenziali non preserva valore di verità

Esempio 8.5.1:

Sia α = "la capitale dell'Italia"

Sia β = "Roma"

Assumiamo $\alpha = \beta$ (stessa estensione)

Contesto: "Giovanni crede che $\square$ sia grande"

$C[\alpha]$: "Giovanni crede che la capitale dell'Italia sia grande" = TRUE

$C[\beta]$: "Giovanni crede che Roma sia grande" = FALSE

(se Giovanni non sa che Roma è la capitale)

Quindi $C[\]$ è opaco.

*Teorema 8.5.2 (Caratterizzazione dei contesti opachi):**

$C[\]$ è opaco $\Leftrightarrow C[\]$ involve intensioni (non solo estensioni)

8.5.3 Principio di Composizionalità Intensionale

*Teorema 8.5.3 (Composizionalità estesa):**

In contesti opachi:

$\llbracket \alpha(\beta) \rrbracket = \llbracket \alpha \rrbracket(\llbracket \beta \rrbracket)$ (α prende l'intensione di β)

Non:

$\llbracket \alpha(\beta) \rrbracket = \llbracket \alpha \rrbracket(\llbracket \beta \rrbracket)$ (α prende l'estensione di β)

Esempio 8.5.2:

$\llbracket \text{crede} \rrbracket: D_e \rightarrow D_{\langle s, t \rangle} \rightarrow D_t$

"John crede che φ ":

$\llbracket \text{crede} \rrbracket(\llbracket \text{John} \rrbracket)(\llbracket \varphi \rrbracket)$

Dove φ è l'intensione (proposizione): $D_{\langle s, t \rangle}$

8.6 Trattamento dei Tempi Verbali

8.6.1 Logica Temporale in IL

*Definizione 8.6.1 (Operatori temporali):**

P (Past): φ era vero

F (Future): ϕ sarà vero

H (Has always been): ϕ è sempre stato vero

G (will always be): ϕ sarà sempre vero

Semantica:

$$\llbracket P \phi \rrbracket^{\langle w, t \rangle, g} = 1 \Leftrightarrow \exists t' < t: \llbracket \phi \rrbracket^{\langle w, t' \rangle, g} = 1$$

$$\llbracket F \phi \rrbracket^{\langle w, t \rangle, g} = 1 \Leftrightarrow \exists t' > t: \llbracket \phi \rrbracket^{\langle w, t' \rangle, g} = 1$$

$$\llbracket H \phi \rrbracket^{\langle w, t \rangle, g} = 1 \Leftrightarrow \forall t' < t: \llbracket \phi \rrbracket^{\langle w, t' \rangle, g} = 1$$

$$\llbracket G \phi \rrbracket^{\langle w, t \rangle, g} = 1 \Leftrightarrow \forall t' > t: \llbracket \phi \rrbracket^{\langle w, t' \rangle, g} = 1$$

*Teorema 8.6.1 (Dualità temporale):**

$$P \phi \equiv \neg H \neg \phi$$

$$F \phi \equiv \neg G \neg \phi$$

$$H \phi \equiv \neg P \neg \phi$$

$$G \phi \equiv \neg F \neg \phi$$

8.6.2 Aspetto: Perfettivo vs. Imperfettivo

*Definizione 8.6.2 (Aspetto come operatore):**

Perfettivo (evento completo):

$$\text{PERF}(\phi): \exists \text{ intervallo } I: [t_{\text{start}}, t_{\text{end}}] \subset I \wedge$$

$$\forall t' \in I: \llbracket \phi \rrbracket^{\{t'\}} = 1 \wedge$$

ϕ è "completato" in I

Imperfettivo (evento in corso):

$$\text{IMPERF}(\phi, t): \exists \text{ intervallo } I \text{ esteso: } t \in I \wedge \phi \text{ in progresso in } I$$

*Teorema 8.6.2 (Paradosso imperfettivo - formale):**

$\text{IMPERF}(\varphi) \not\models \text{PERF}(\varphi)$

Esempio:

"John stava attraversando la strada" $\not\models$ "John attraversò la strada"

Dimostrazione: Controesempio: John interrotto a metà $\rightarrow$ imperfettivo TRUE, perfettivo FALSE. ■

8.7 Presupposizioni e Proiezione

8.7.1 Teoria delle Presupposizioni

*Definizione 8.7.1 (Presupposizione):**

φ presuppone ψ ($\varphi \triangleright \psi$) $\Leftrightarrow$

$[[\varphi]]$ è definita solo se $[[\psi]] = 1$

Formalmente (logica a tre valori):

$[[\varphi]] \in \{0, 1\} \Leftrightarrow [[\psi]] = 1$

$[[\varphi]] = \# \text{ (undefined)} \Leftrightarrow [[\psi]] \neq 1$

Esempio 8.7.1:

"Il re di Francia è calvo" presuppone "La Francia ha un re"

$[[\text{il re di Francia è calvo}]] =$

1 se $\exists !x[\text{re-di-Francia}(x) \wedge \text{calvo}(x)]$

0 se $\exists !x[\text{re-di-Francia}(x) \wedge \neg \text{calvo}(x)]$

se $\neg \exists !x[\text{re-di-Francia}(x)]$

8.7.2 Proiezione delle Presupposizioni

*Definizione 8.7.2 (Famiglia di presupposizioni - Holes, Plugs, Filters):**

Holes (pass presuppositions through):

Esempio: "non", "crede"

Se $\phi \triangleright \psi$, allora "non ϕ " $\triangleright \psi$, "x crede ϕ " $\triangleright \psi$

Plugs (block presuppositions):

Esempio: "dire che"

Se $\phi \triangleright \psi$, "x dice che ϕ " $\ntriangleright \psi$

Filters (conditional projection):

Esempio: "se... allora"

Se $\phi \triangleright \psi$, allora "se α allora ϕ " $\triangleright (\alpha \rightarrow \psi)$

*Teorema 8.7.1 (Algoritmo di proiezione):**

Per congiunzione:

$(\phi \wedge \psi) \triangleright (\triangleright(\phi) \cup (\phi \rightarrow \triangleright(\psi)))$

Per condizionale:

$(\phi \rightarrow \psi) \triangleright (\triangleright(\phi) \cup (\phi \rightarrow \triangleright(\psi)))$

8.8 PTQ: Analisi Completa

8.8.1 Frammento PTQ - Sistema Formale

Sintassi di PTQ:

Categorie base:

t (term), IV (intransitive verb), TV (transitive verb),

CN (common noun), Det (determiner), S (sentence)

Regole sintattiche (selezione):

S1: $t, IV \rightarrow S: F_1(\alpha, \delta) = \alpha \delta$

S2: $TV, t \rightarrow IV: F_2(\gamma, \beta) = \gamma \beta$

S3: $Det, CN \rightarrow t: F_3(\delta, \xi) = \delta \xi$

S4: $CN, IV \rightarrow CN: F_4(\xi, \delta) = \xi \text{ tale che } \delta$

...

(17 regole totali in PTQ)

Traduzioni in IL:

Per ogni regola F_n , una translation rule T_n che mappa sintassi a IL.

Esempio 8.8.1 (Traduzione di S1):

Sintassi: $F_1(\text{John}, \text{walk}) = \text{"John walk"}$

Translation: $T_1(\text{john}', \text{walk}') = \text{walk}'(\text{john}')$

dove:

john' : e

walk' : $\langle e, t \rangle$

8.8.2 Quantificatori in PTQ

*Definizione 8.8.1 (Type-raising per termini):**

Basic terms (tipo e) $\rightarrow$ Quantified terms (tipo $\langle \langle e, t \rangle, t \rangle$)

Algoritmo di type-raising:

$\text{term}_e \rightarrow \lambda P \langle e, t \rangle. P(\text{term}_e)$

Esempio:

john: e

→ $\lambda P. P(\text{john}): \langle \langle e, t \rangle, t \rangle$

*Teorema 8.8.1 (Uniformità del trattamento):**

In PTQ, tutti i termini (nomi propri, quantificatori, descrizioni)

hanno lo stesso tipo: $\langle \langle e, t \rangle, t \rangle$

Vantaggi:

1. Compositionalità uniforme
2. Scope ambiguity naturale
3. Trattamento sistematico di de re/de dicto

8.8.3 Esempio Completo PTQ

Derivazione di "Ogni uomo ama una donna":

Step 1: Lessico

ogni: $\langle \langle e, t \rangle, \langle \langle e, t \rangle, t \rangle \rangle$

uomo: $\langle e, t \rangle$

ama: $\langle e, \langle e, t \rangle \rangle$

una: $\langle \langle e, t \rangle, \langle \langle e, t \rangle, t \rangle \rangle$

donna: $\langle e, t \rangle$

Step 2: Applicazioni sintattiche

[ogni uomo]: $\langle \langle e, t \rangle, t \rangle$

[una donna]: $\langle \langle e, t \rangle, t \rangle$

[ama una donna]: $\langle e, t \rangle$

[ogni uomo ama una donna]: t

Step 3: Traduzioni in IL

$$[[\text{ogni}]](A)(B) = \forall x[A(x) \rightarrow B(x)]$$

$$[[\text{una}]](A)(B) = \exists x[A(x) \wedge B(x)]$$

$$[[\text{ama}]](x)(y) = \text{ama}'(y)(x)$$

Step 4: Composizione

$$LF_1 (\forall > \exists): [[\text{ogni}]]([[\text{uomo}]])(\lambda x. [[\text{una}]]([[\text{donna}]])(\lambda y. [[\text{ama}]](x)(y)))$$

$$= \forall x[\text{uomo}'(x) \rightarrow \exists y[\text{donna}'(y) \wedge \text{ama}'(y)(x)]]$$

$$LF_2 (\exists > \forall): [[\text{una}]]([[\text{donna}]])(\lambda y. [[\text{ogni}]]([[\text{uomo}]])(\lambda x. [[\text{ama}]](x)(y)))$$

$$= \exists y[\text{donna}'(y) \wedge \forall x[\text{uomo}'(x) \rightarrow \text{ama}'(y)(x)]]$$

Parte II: Dynamic Semantics, DRT ed Estensioni Moderne

8.9 Dynamic Semantics: Oltre la Composizionalità Statica

8.9.1 Motivazioni per la Semantica Dinamica

La semantica di Montague è **statica**: il significato di una frase è una funzione da indici a valori di verità, e non c'è "memoria" del contesto precedente.

Problema fondamentale: Anafora inter-frasale

Esempio 8.9.1 (Fallimento della semantica statica):

(1) Un uomo_i entrò. Egli_i fischiava.

Analisi statica (fallisce):

- Frase 1: $\exists x[\text{uomo}(x) \wedge \text{entrò}(x)]$

- Frase 2: fischiava(???)

Il pronome "egli" non può essere legato dal quantificatore della frase precedente!

Teorema 8.9.1 (Limite della composizionalità statica):

In semantica statica con composizionalità standard:

$$\llbracket \varphi_1 \cdot \varphi_2 \rrbracket = \llbracket \varphi_1 \rrbracket \wedge \llbracket \varphi_2 \rrbracket$$

Ma questo non permette che φ_1 introduca variabili accessibili in φ_2 .

8.9.2 Svolta Dinamica: Cambiamento di Paradigma

Definizione 8.9.1 (Semantica Dinamica - idea centrale):

Il significato di una frase non è più un valore di verità, ma una **relazione tra stati di informazione**:

$$\llbracket \varphi \rrbracket: \text{Context} \rightarrow \wp(\text{Context})$$

Dove Context è uno stato informativo (assignment, discourse state, etc.)

Principio chiave: Le frasi **cambiano** il contesto, non solo lo valutano.

8.9.3 Dynamic Predicate Logic (DPL) - Formalizzazione

Definizione 8.9.2 (Sistema DPL - Groenendijk & Stokhof 1991):

Sintassi: come logica del primo ordine

Semantica:

$$\llbracket \cdot \rrbracket: \text{Formula} \rightarrow (\text{Assignment} \rightarrow \wp(\text{Assignment}))$$

Un'assignment $g: \text{Var} \rightarrow D$ mappa variabili a entità.

Clausole semantiche fondamentali:

Atomiche:

$$\begin{aligned} \llbracket P(t_1, \dots, t_n) \rrbracket(g) &= \{g\} \text{ se } \langle \llbracket t_1 \rrbracket^g, \dots, \llbracket t_n \rrbracket^g \rangle \in \llbracket P \rrbracket \\ &= \emptyset \text{ altrimenti} \end{aligned}$$

Congiunzione (dinamica!):

$$\llbracket \varphi \wedge \psi \rrbracket(g) = \bigcup \{ \llbracket \varphi \rrbracket(g) \cap \llbracket \psi \rrbracket(h) \mid g \sqsubseteq h \}$$

Composizione relazionale: ψ è valutata rispetto a *tutti* gli output di φ .

Negazione (statica!):

$$\begin{aligned} \llbracket \neg \varphi \rrbracket(g) &= \{g\} \text{ se } \llbracket \varphi \rrbracket(g) = \emptyset \\ &= \emptyset \text{ altrimenti} \end{aligned}$$

La negazione "reseta": non passa modifiche al contesto.

Quantificatore esistenziale (dinamico!):

$$\llbracket \exists x \varphi \rrbracket(g) = \bigcup_{a \in D} \llbracket \varphi \rrbracket(g[x:=d])$$

Dove $g[x:=d]$ è come g eccetto che $g[x:=d] = d$.

Quantificatore universale (statico!):

$$\begin{aligned} \llbracket \forall x \varphi \rrbracket(g) &= \{g\} \text{ se } \forall d \in D: \llbracket \varphi \rrbracket(g[x:=d]) \neq \emptyset \\ &= \emptyset \text{ altrimenti} \end{aligned}$$

8.9.4 Teoremi Fondamentali di DPL**Teorema 8.9.2 (Binding dinamico):**

In DPL:

$$\llbracket \exists x (P(x) \wedge Q(x)) \rrbracket \neq \llbracket \exists x P(x) \wedge Q(x) \rrbracket \text{ in generale}$$

Ma entrambi permettono binding! La differenza è nello scope.

Teorema 8.9.3 (Asimmetria della congiunzione):

$$\llbracket \varphi \wedge \psi \rrbracket \neq \llbracket \psi \wedge \varphi \rrbracket \text{ (in generale)}$$

Esempio:

$$\llbracket \exists x P(x) \wedge Q(x) \rrbracket \neq \llbracket Q(x) \wedge \exists x P(x) \rrbracket$$

La prima vincola x , la seconda no (x libera in $Q(x)$).

Teorema 8.9.4 (Operators statico/dinamico):

Operatore	Tipo	Passa binding?
$\wedge$	Dinamico	Sì (da sinistra)
$\vee$	Statico	No
$\neg$	Statico	No
$\exists$	Dinamico	Sì
$\forall$	Statico	No

Dimostrazione ($\forall$ è statico):

Consideriamo: $\llbracket \forall x P(x) \wedge Q(x) \rrbracket$

$$\llbracket \forall x P(x) \wedge Q(x) \rrbracket(g) = \bigcup \square \in \llbracket \forall x P(x) \rrbracket(g) \llbracket Q(x) \rrbracket(h)$$

Ma $\llbracket \forall x P(x) \rrbracket(g) = \{g\}$ o $\emptyset$, quindi:

- Se successo, $h = g$ (nessuna modifica!)
- Quindi x in $Q(x)$ ha lo stesso valore che aveva in g

Il quantificatore universale non estende il binding. ■

8.9.5 Risoluzione del Problema dell'Anafora

Esempio 8.9.2 (Successo di DPL):

"Un uomo entrò. Egli fischiava."

Rappresentazione:

$$\exists x(\text{uomo}(x) \wedge \text{entrò}(x)) \wedge \text{fischia}(x)$$

Derivazione:

$$\begin{aligned} & \llbracket \exists x(\text{uomo}(x) \wedge \text{entrò}(x)) \wedge \text{fischia}(x) \rrbracket(g_0) \\ &= \bigcup \square \in \llbracket \exists x(\text{uomo}(x) \wedge \text{entrò}(x)) \rrbracket(g_0) \llbracket \text{fischia}(x) \rrbracket(h) \end{aligned}$$

Il quantificatore $\exists x$ modifica l'assignment:

$$\llbracket \exists x(\text{uomo}(x) \wedge \text{entrò}(x)) \rrbracket(g_0) = \{g_1, g_2, \dots\}$$

dove ogni $g_i(x)$ è un uomo che è entrato.

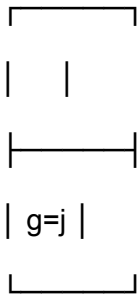
Quindi $\text{fischia}(x)$ è valutato con x già vincolato!

8.9.6 Confronto: Semantica Statica vs Dinamica

Tabella comparativa:

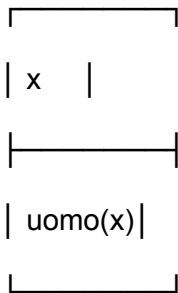
Aspetto	Statica (Montague)	Dinamica (DPL)
Tipo semantico	$\varphi \rightarrow \{0,1\}$	$\varphi: g \rightarrow \wp(g)$
Composizionalità	$\llbracket \varphi \wedge \psi \rrbracket = \llbracket \varphi \rrbracket \wedge \llbracket \psi \rrbracket$	$\llbracket \varphi \wedge \psi \rrbracket(g) = \bigcup \llbracket \psi \rrbracket(h)$
Congiunzione	Simmetrica	Asimmetrica
Binding	Intra-frasale	Inter-frasale

"Giovanni" $\rightarrow$



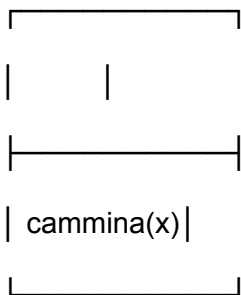
R2: Descrizione indefinita

"un uomo" $\rightarrow$



R3: Verbo intransitivo

"cammina" + referente x $\rightarrow$



R4: Merge (composizione di frasi)

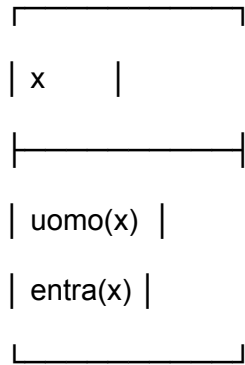
Se $K_1 = \langle U_1, Con_1 \rangle$ e $K_2 = \langle U_2, Con_2 \rangle$, allora:

$$K_1 + K_2 = \langle U_1 \cup U_2, Con_1 \cup Con_2 \rangle$$

8.10.3 Esempio Completo di Costruzione

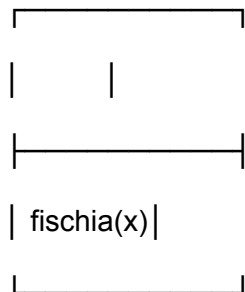
Esempio 8.10.1: "Un uomo entra. Egli fischia."

Step 1: Frase 1: "Un uomo entra"

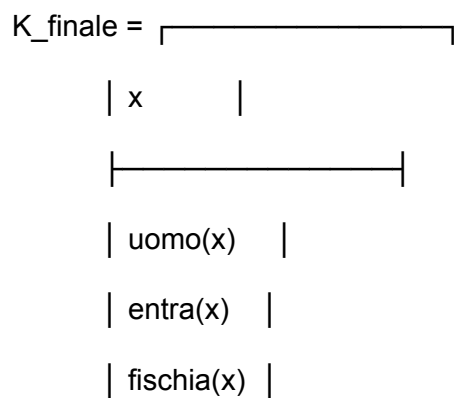


Step 2: Frase 2: "Egli fischia"

Il pronome "egli" risolve a x già disponibile:



Step 3: Merge



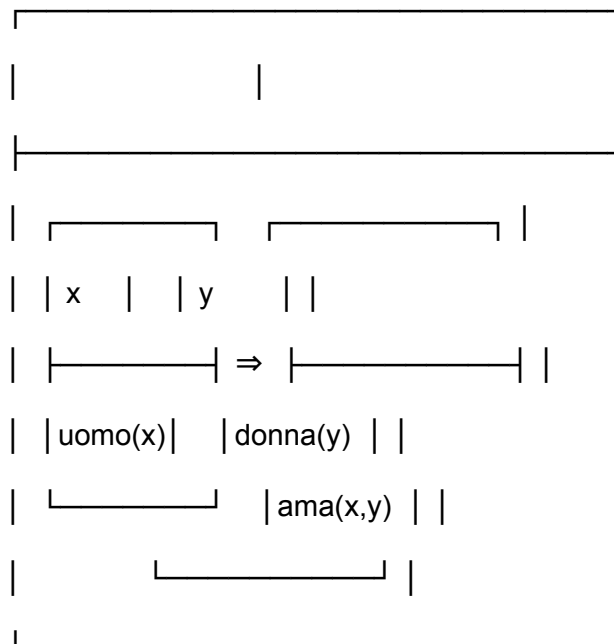
8.10.4 Condizioni Complesse e DRS Embedded

Definizione 8.10.3 (Condizioni complesse):

Le condizioni possono essere:

1. Atomiche: $P(x_1, \dots, x_n)$
2. Negazione: $\neg K$
3. Implicazione: $K_1 \Rightarrow K_2$
4. Disgiunzione: $K_1 \vee K_2$

Esempio 8.10.2: "Ogni uomo ama una donna"



8.10.5 Accessibilità dei Referenti

Definizione 8.10.4 (Relazione di accessibilità):

Un discourse referent x è **accessibile** in una DRS K se:

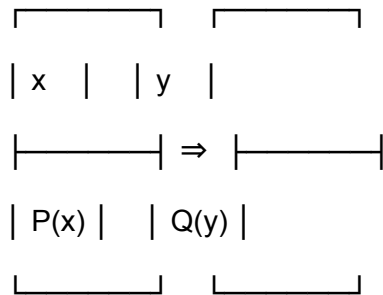
1. $x \in U(K)$, oppure
2. x è accessibile in una DRS K' che domina K

Regole di dominanza:

- In $K_1 \Rightarrow K_2$: referenti di K_1 sono accessibili in K_2

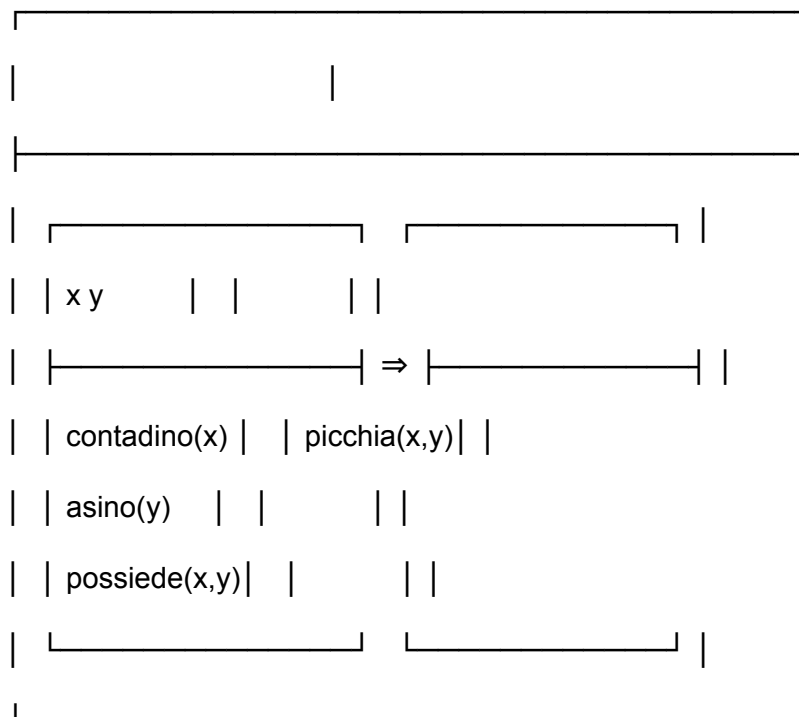
- In $\neg K$: referenti di K NON sono accessibili fuori
- In $K_1 \vee K_2$: referenti interni non sono accessibili fuori

Teorema 8.10.1 (Asimmetria dell'implicazione):



Qui: x accessibile in secondo box, ma y NON accessibile fuori dall'implicazione.

Esempio 8.10.3: "Se un contadino possiede un asino, lo picchia."



x, y accessibili nel conseguente!

8.10.6 Semantica Formale delle DRS

Definizione 8.10.5 (Embedding function):

Una funzione di embedding f per una DRS $K = \langle U, \text{Con} \rangle$ in un modello M è:

$f: U \rightarrow D_M$

tale che tutte le condizioni in Con sono soddisfatte.

Definizione 8.10.6 (Verità di una DRS):

K è vera in M ($M \models K$) $\Leftrightarrow$ esiste un embedding f per K in M

Formalmente:

$M \models_f K \Leftrightarrow \forall \varphi \in \text{Con}: M \models_f \varphi$

dove per condizioni atomiche:

$M \models_f P(x_1, \dots, x_n) \Leftrightarrow \langle f(x_1), \dots, f(x_n) \rangle \in \llbracket P \rrbracket^M$

8.10.7 Condizioni Complesse - Semantica

Negazione:

$M \models_f \neg K \Leftrightarrow \nexists g$ che estende f : $M \models_g K$

Implicazione:

$M \models_f (K_1 \Rightarrow K_2) \Leftrightarrow \forall g$ che estende f :

se $M \models_g K_1$ allora $\exists h$ che estende g : $M \models_h K_2$

Disgiunzione:

$M \models_f (K_1 \vee K_2) \Leftrightarrow \exists g$ che estende f : $M \models_g K_1$ oppure $M \models_g K_2$

8.10.8 Teoremi di Equivalenza

Teorema 8.10.2 (Traduzione DRS $\rightarrow$ FOL):

Ogni DRS può essere tradotta in una formula del primo ordine:

$\text{tr}(\langle \{x_1, \dots, x_n\}, \{\varphi_1, \dots, \varphi_m\} \rangle) = \exists x_1 \dots \exists x_n (\varphi_1 \wedge \dots \wedge \varphi_m)$

con ricorsione per condizioni complesse:

$$\text{tr}(\neg K) = \neg \text{tr}(K)$$

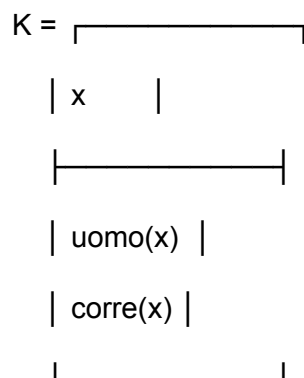
$$\text{tr}(K_1 \Rightarrow K_2) = \forall \vec{x}_1 (\text{tr_open}(K_1) \rightarrow \text{tr}(K_2))$$

dove $\vec{x}_1$ sono i referenti introdotti in K_1 .

Teorema 8.10.3 (Correttezza della traduzione):

$$M \models K \Leftrightarrow M \models \text{tr}(K)$$

Esempio 8.10.4:



$$\text{tr}(K) = \exists x (\text{uomo}(x) \wedge \text{corre}(x))$$

8.10.9 Vantaggi di DRT rispetto a FOL

1. Rappresentazione strutturata del discorso

DRT mantiene la struttura incrementale del discorso, FOL no.

2. Anafora naturale

Binding automatico attraverso accessibilità, non scope sintattico.

3. Rappresentazione cognitiva

Le DRS box riflettono meglio la costruzione mentale del significato.

Teorema 8.10.4 (Potere espressivo):

DRT e FOL hanno lo stesso potere espressivo per frasi isolate, ma DRT è più espressiva per:

- Discorsi multi-frase
- Anafora complessa
- Presupposizioni (estensioni)

8.11 File Change Semantics e Update Semantics**8.11.1 File Change Semantics (FCS) - Heim 1982****Definizione 8.11.1 (File come contesto):**

Un **file** f è un insieme di possibilità (worlds/assignments):

$$f \subseteq W \times G$$

dove W = mondi possibili, G = assignments parziali

Metafora: Un file è come una cartella con "carte" (file cards), ognuna rappresenta un'entità introdotta nel discorso.

8.11.2 Operazioni sui Files**Definizione 8.11.2 (Aggiornamento del file):**

Il significato di una frase φ è una funzione di update:

$$[[\varphi]]: \wp(W \times G) \rightarrow \wp(W \times G)$$

Interpretazione: φ cambia lo stato informativo da f a f' .

Update conditions principali:**1. Predicazione:**

$$f[P(x)] = \{\langle w, g \rangle \in f : g(x) \in [[P]]^w\}$$

Filtra via le possibilità dove x non è P .

2. Introduzione di referente:

$$f[\exists x] = \{\langle w, g[x:=d] \rangle : \langle w, g \rangle \in f, d \in D^w\}$$

Estende ogni possibilità con un nuovo valore per x.

3. Congiunzione:

$$f[\varphi \wedge \psi] = (f[\varphi])[\psi]$$

Composizione sequenziale: prima φ , poi ψ sull'output.

4. Negazione:

$$f[\neg\varphi] = f \text{ se } f[\varphi] = \emptyset$$

$$= \emptyset \text{ altrimenti}$$

8.11.3 Esempio Completo FCS

Esempio 8.11.1: "Un gatto entra. Il gatto miagola."

Step 1: f_0 = stato iniziale (file vuoto rispetto a gatti)

Step 2: "Un gatto entra"

$$f_1 = f_0[\exists x][\text{gatto}(x)][\text{entra}(x)]$$

= introduce x, filtra per $\text{gatto}(x)$, filtra per $\text{entra}(x)$

Step 3: "Il gatto miagola"

Articolo determinativo "il" richiede:

- x già nel file (familiarità)
- x unico gatto saliente (unicità)

$$f_2 = f_1[\text{miagola}(x)]$$

= filtra f_1 per $\text{miagola}(x)$

8.11.4 Presupposizioni in FCS

Definizione 8.11.3 (Presupposizione come condizione sul file):

Una frase φ presuppone ψ se:

$f[\varphi]$ è definito solo se f soddisfa ψ

Esempio 8.11.2: Descrizioni definite

"Il re di Francia è calvo"

Presuppone: f contiene esattamente un referente x con $\text{re-di-Francia}(x)$

$f[\text{il } x: \text{re-di-Francia}(x)][\text{calvo}(x)]$ definito $\Leftrightarrow$

$\exists ! \langle w, g \rangle \in f$ con $g(x) \in \llbracket \text{re-di-Francia} \rrbracket^w$

8.11.5 Confronto: FCS vs DRT vs DPL

Tabella comparativa:

Aspetto	DRT	DPL	FCS
Oggetto base	DRS box	Assignment	File (set di $\langle w, g \rangle$)
Composizione	Merge	Relazione	Update sequenziale
Presupposizioni	Estensione	Non native	Native
Anafora	Accessibilità	Binding dinamico	File cards
Notazione	Grafica	Logica	Set-teorica

Teorema 8.11.1 (Equivalenza empirica):

Per discorsi senza presupposizioni:

$\llbracket \varphi \rrbracket^{\text{DRT}}, \llbracket \varphi \rrbracket^{\text{DPL}}, \llbracket \varphi \rrbracket^{\text{fcs}}$ sono inter-traducibili

8.11.6 Update Semantics - Veltman 1996

Definizione 8.11.4 (Stati epistemici):

Uno stato s è un insieme di mondi possibili (informazione non massimamente specifica):

$$s \subseteq W$$

Update come test e espansione:

$$s[\varphi] = \{w \in s : \llbracket \varphi \rrbracket^w = 1\} \text{ (per assertive)}$$

Definizione 8.11.5 (Modalità epistemiche):

Might: $\Diamond\varphi$

$$\begin{aligned} s[\Diamond\varphi] &= s \text{ se } s[\varphi] \neq \emptyset \\ &= \emptyset \text{ altrimenti} \end{aligned}$$

"Might φ " testa se φ è compatibile con s .

Must: $\Box\varphi$

$$\begin{aligned} s[\Box\varphi] &= s \text{ se } s \subseteq \llbracket \varphi \rrbracket \\ &= \emptyset \text{ altrimenti} \end{aligned}$$

"Must φ " testa se φ è necessario dato s .

8.11.7 Sequenze di Updates

Esempio 8.11.3: Ragionamento epistemico

Discorso: "Potrebbe piovere. Ma non piove. Quindi deve essere nuvoloso."

$$s_0 = \{w_1, w_2, w_3, w_4\} \text{ (tutti i mondi possibili rilevanti)}$$

$$s_1 = s_0[\Diamond\text{piove}]$$

$= s_0$ (test: piove compatibile)

$s_2 = s_1[\neg \text{piove}]$

$= \{w \in s_1 : \neg \text{piove}(w)\}$

$= \{w_1, w_3\}$ (elimina mondi dove piove)

$s_3 = s_2[\Box \text{nuvoloso}]$

$= s_2$ se $\forall w \in s_2: \text{nuvoloso}(w)$

8.11.8 Logica dell'Update

Teorema 8.11.2 (Leggi dell'update):

Non sono tutte classiche!

1. Congiunzione:

$s[\varphi \wedge \psi] = s[\varphi][\psi]$ ✓ (sequenziale)

2. Idempotenza (fallisce!):

$s[\varphi][\varphi] \neq s[\varphi]$ in generale

Controesempio con $\Diamond$:

$s[\Diamond \varphi][\Diamond \varphi] = s[\Diamond \varphi]$ (secondo test ridondante)

$s[\Diamond \varphi]$ potrebbe essere diverso da $s[\Diamond \varphi][\Diamond \varphi]$

3. Assorbimento (vale!):

$s[\varphi][\varphi \wedge \psi] = s[\varphi][\psi]$

4. Non-contraddizione:

$$s[\varphi][\neg\varphi] = \emptyset$$

8.11.9 Test vs Eliminativi

Definizione 8.11.6 (Classificazione degli operatori):

Eliminativi: riducono sempre lo stato

$$s[\varphi] \subseteq s$$

Esempi: asserzioni, $\neg\varphi$

Test: preservano o annullano

$$s[\varphi] = s \text{ oppure } s[\varphi] = \emptyset$$

Esempi: $\Diamond\varphi$, $\Box\varphi$

Teorema 8.11.3 (Caratterizzazione):

Un operatore O è un test $\Leftrightarrow$

$$s[O(\varphi)] = s \text{ oppure } s[O(\varphi)] = \emptyset$$

e inoltre:

$$s[O(\varphi)][O(\varphi)] = s[O(\varphi)] \quad (\text{idempotenza})$$

8.11.10 Applicazioni: Condizionali e Default Reasoning

Esempio 8.11.4: Condizionali epistemic

"Se Gianni è a casa, allora sta dormendo"

Interpretazione update:

$$s[\text{if } \varphi \text{ then } \psi] = s \text{ se } s[\varphi] \subseteq \llbracket \psi \rrbracket$$

$$= \emptyset \text{ altrimenti}$$

Ragionamento nonmonotono:

$$S_0 = \{W_1, W_2, W_3\}$$

$$S_1 = S_0[\text{gli uccelli volano}] \text{ (default)}$$

= filtra per volare

$$S_2 = S_1[\text{Tweety è un pinguino}] \text{ (eccezione)}$$

= deve rivedere!

Update semantics cattura naturalmente revisione delle credenze.

8.12 Estensioni Moderne della Semantica di Montague

8.12.1 Type-Theoretical Grammar (TTG)

Motivazione: Montague usa lambda calcolo simply-typed. Le teorie dei tipi moderne (Martin-Löf, CoC) offrono maggiore espressività.

Definizione 8.12.1 (Dependent Types in Semantics):

In TTG, i tipi possono dipendere da valori:

$$\prod x:A. B(x) \text{ (tipo dipendente)}$$

Esempio 8.12.1: Predicati con lunghezza fissata

$$\text{Vector}: \mathbb{N} \rightarrow \text{Type}$$

$$\text{Vector}(0) = \text{Unit}$$

$$\text{Vector}(n+1) = \mathbb{R} \times \text{Vector}(n)$$

Applicazione linguistica:

$$\text{lista-di-n-elementi}: \prod n:\mathbb{N}. \text{List}(\text{Entity}, n)$$

8.12.2 Proposizioni come Tipi

Principio di Curry-Howard esteso alla semantica:

Proposizioni $\approx$ Tipi

Prove $\approx$ Termini

Definizione 8.12.2 (Meaning as proof):

Il significato di " φ è vero" = un termine di tipo φ :

$\llbracket \varphi \rrbracket : \varphi$

Esempio 8.12.2: Anafora come dipendenza

"Un uomo cammina. Egli fischia."

$\Sigma x:\text{Man. (Walk}(x) \times \text{Whistle}(x))$

Tipo Sigma (somma dipendente): esiste x di tipo Man, e abbiamo prove di Walk(x) E Whistle(x).

8.12.3 Inductive Types per Plurali e Eventi

Definizione 8.12.3 (Link structure - Landman 1989):

Inductive Entity:

| atom: Individual $\rightarrow$ Entity

| join: Entity $\rightarrow$ Entity $\rightarrow$ Entity

Assiomi:

$\text{join}(a,b) = \text{join}(b,a)$ (commutatività)

$\text{join}(a,\text{join}(b,c)) = \text{join}(\text{join}(a,b),c)$ (associatività)

$\text{join}(a,a) = a$ (idempotenza)

Esempio 8.12.3: Plurali

"Tre uomini sollevano un pianoforte"

$\exists x:\text{Entity}. |\text{atomic_parts}(x)| = 3 \wedge$

$\forall y \in \text{atomic_parts}(x). \text{uomo}(y) \wedge$

$\text{sollevano}(x, \text{pianoforte})$

8.12.4 Glue Semantics (LFG)

Definizione 8.12.4 (Linear Logic per Composizione):

Glue usa logica lineare per assemblare significati:

$A \multimap B$ (implicazione lineare: "consuma" A per produrre B)

Lessico in Glue:

uomo: $\text{uomo}'(x) : gx \multimap gx$

ogni: $\forall X, Y. (gx \multimap X) \multimap (gx \multimap Y) \multimap (X \multimap Y) : h$

dove g, h sono "glue variables" (tipi di significati).

Esempio 8.12.4: "Ogni uomo corre"

Derivazione in linear logic:

ogni: $(gx \multimap X) \multimap (gx \multimap Y) \multimap (X \multimap Y)$

uomo: $gx \multimap gx$

corre: $gx \multimap t$

Combinazione:

ogni @ $(\lambda x. \text{uomo}(x)) @ (\lambda y. \text{corre}(y)) : t$

$= \forall x[\text{uomo}(x) \rightarrow \text{corre}(x)]$

8.12.5 Continuations in Semantics

Definizione 8.12.5 (Continuation-Passing Style - CPS):

Invece di:

$\llbracket \varphi \rrbracket: t$

Usiamo:

$\llbracket \varphi \rrbracket: (t \rightarrow \alpha) \rightarrow \alpha$ (doppia negazione)

Vantaggio: Modella scope, focus, e side-effects.

Esempio 8.12.5: Quantifier scope con continuations

$\llbracket \text{ogni uomo} \rrbracket^{\text{cps}} = \lambda k. \forall x[\text{uomo}(x) \rightarrow k(x)]$

$\llbracket \text{qualche donna} \rrbracket^{\text{cps}} = \lambda k. \exists y[\text{donna}(y) \wedge k(y)]$

Combinazione:

$\llbracket \text{ogni uomo ama qualche donna} \rrbracket$

$= \llbracket \text{ogni uomo} \rrbracket(\lambda x. \llbracket \text{qualche donna} \rrbracket(\lambda y. \text{ama}(x,y)))$

$= \lambda k. \forall x[\text{uomo}(x) \rightarrow \exists y[\text{donna}(y) \wedge k(\text{ama}(x,y))]]$

Scope automaticamente $\forall > \exists$.

Per invertire scope, usiamo continuation operators (type-shifting).

8.12.6 Hamblin Semantics per Alternative

Definizione 8.12.6 (Sets of alternatives):

Il significato di una frase con focus/questioni è un insieme di alternative:

$\llbracket \varphi \rrbracket^{\text{Hamblin}} \subseteq D_{\text{T}}$

Esempio 8.12.6: Domande wh-

"Chi corre?"

$\llbracket \text{chi corre} \rrbracket = \{\text{corre}(d) : d \in D_e\}$

= insieme di proposizioni

Focus:

"GIOVANNI corre" (focus su Giovanni)

$\llbracket \text{GIOVANNI}_F \text{ corre} \rrbracket = \{\text{corre}(x) : x \in D_e\}$

8.12.7 Inquisitive Semantics

Definizione 8.12.7 (Proposizioni come inquisitive):

Una proposizione è un insieme di insiemi di mondi (downward closed):

$$\llbracket \varphi \rrbracket \subseteq \wp(W)$$

Interpretazione:

- Ogni $s \in \llbracket \varphi \rrbracket$ è un modo di risolvere l'issue di φ
- Se $\llbracket \varphi \rrbracket = \{W\}$, φ è puramente informativa
- Se $|\llbracket \varphi \rrbracket| > 1$, φ è inquisitiva (pone questione)

Esempio 8.12.7:

$\llbracket \text{piove} \rrbracket = \{\{w : \text{piove}(w)\}, \{w : \neg \text{piove}(w)\}\}$ (non-inquisitivo)

$\llbracket \text{piove?} \rrbracket = \{\{w : \text{piove}(w)\}, \{w : \neg \text{piove}(w)\}\}$ (inquisitivo)

Notare: stessa semantica! Ma pragmaticamente diversi.

Disgiunzione inquisitiva:

$$\llbracket \varphi \vee \psi \rrbracket = \llbracket \varphi \rrbracket \cup \llbracket \psi \rrbracket$$

"Piove o nevica?" = modo di chiedere quale dei due.

8.12.8 Degree Semantics - Comparativi e Superlativi

Definizione 8.12.8 (Degrees come ordini):

Degree = $\langle D_d, \leq \rangle$ dove D_d è un insieme ordinato

Aggettivi graduali:

$[[\text{alto}]]: D_e \rightarrow D_d \rightarrow D_t$

$[[\text{alto}]](x)(d) = 1 \Leftrightarrow \text{altezza}(x) \geq d$

Forma positiva (threshold):

$[[\text{alto}]]_{\text{pos}}(x) = \exists d [\text{alto}(x, d) \wedge d > \theta_{\text{alto}}(C)]$

dove $\theta_{\text{alto}}(C)$ = threshold contestuale.

Comparativo:

$[[\text{più alto di}]](x)(y) = \exists d_1, d_2 [\text{alto}(x, d_1) \wedge \text{alto}(y, d_2) \wedge d_1 > d_2]$

Superlativo:

$[[\text{il più alto}]](x)(C) = \forall y \in C [y \neq x \rightarrow \text{più-alto}(x, y)]$

8.12.9 Event Semantics - Davidson e Neo-Davidsoniane**Definizione 8.12.9 (Eventi come oggetti):**

Oltre a individui, introduciamo eventi:

$D_e \uplus D_v$ (v = events)

Verbi introducono variabili di evento:

$[[\text{corre}]] = \lambda x. \lambda e. \text{correre}(e) \wedge \text{agente}(e, x)$

Vantaggi:

1. Modificazione avverbiale naturale

$[[\text{correre velocemente}]] = \lambda x. \lambda e. \text{correre}(e) \wedge \text{agente}(e, x) \wedge \text{veloce}(e)$

2. Causatività

$[[\text{uccidere}]] = \lambda x. \lambda y. \lambda e. \exists e' [\text{causare}(e, e') \wedge \text{morire}(e') \wedge \text{paziente}(e', x) \wedge \text{agente}(e, y)]$

8.12.10 Mereologia degli Eventi

Definizione 8.12.10 (Part structure):

$e_1 \sqsubseteq e_2$ (e_1 è parte di e_2)

Assiomi:

- Riflessività: $e \sqsubseteq e$
- Transitività: $e_1 \sqsubseteq e_2 \wedge e_2 \sqsubseteq e_3 \rightarrow e_1 \sqsubseteq e_3$
- Antisimmetria: $e_1 \sqsubseteq e_2 \wedge e_2 \sqsubseteq e_1 \rightarrow e_1 = e_2$

Somma di eventi:

$e_1 \sqcup e_2$ = il minimo evento che contiene e_1 e e_2

Teorema 8.12.1 (Plurali come somme):

$[[\text{tre uomini corrono}]] = \exists x, e [|\text{atom}(x)| = 3 \wedge \forall y \in \text{atom}(x). \text{uomo}(y) \wedge \text{correre}(e) \wedge \text{agente}(e, x)]$

dove e può essere somma di sotto-eventi.

8.13 Grammatica di Montague: Approfondimenti Formali

8.13.1 Il Sistema PTQ Completo - Le 17 Regole

Definizione 8.13.1 (Categorie sintattiche PTQ):

$Cat = \{t, e, CN, TV, IV, IAV, T, DET, \dots\}$

dove:

- t = frase (truth-value)
- e = termine (entity)
- CN = nome comune
- TV = verbo transitivo
- IV = verbo intransitivo
- IAV = avverbio intransitivo
- T = tempo verbale
- DET = determinante

8.13.2 Regole Sintattiche (F) e Traduzioni (T)

Regola S0: Subject-Predicate

$F_0(\alpha, \delta) = \alpha \delta$ dove $\alpha \in PT_e$, $\delta \in P_IV$

$T_0(\alpha', \delta') = \delta'(\alpha')$

Esempio: $F_0(\text{John}, \text{walk}) = \text{"John walks"}$

$T_0(j, \text{walk}') = \text{walk}'(j)$

Regola S1: Verbo transitivo + oggetto

$F_1(\beta, \gamma) = \gamma \beta$ dove $\beta \in PT_e$, $\gamma \in P_TV$

$T_1(\beta', \gamma') = \lambda x. \gamma'(\beta')(x)$

Esempio: $F_1(\text{Mary}, \text{find}) = \text{"find Mary"}$

$T_1(m', \text{find}') = \lambda x. \text{find}'(m')(x)$

Regola S2: Determinante + nome comune

$F_2(\zeta, \xi) = \zeta \xi$ dove $\zeta \in P_DET$, $\xi \in P_CN$

$T_2(\zeta', \xi') = \zeta'(\xi')$

Esempio: $F_2(\text{every}, \text{man}) = \text{"every man"}$

$$T_2(\text{every}', \text{man}') = \lambda P. \forall x[\text{man}'(x) \rightarrow P(x)]$$

Regola S3: Relative clause

$$F_3(\xi, \delta) = \xi \text{ such-that } \delta \text{ dove } \xi \in P_CN, \delta \in P_IV$$

$$T_3(\xi', \delta') = \lambda x. \xi'(x) \wedge \delta'(x)$$

Esempio: $F_3(\text{man}, \text{walk}) = \text{"man such that walk"}$

$$T_3(\text{man}', \text{walk}') = \lambda x. \text{man}'(x) \wedge \text{walk}'(x)$$

Regola S4: Intensionalizzazione del verbo

$$F_4(\varphi, \delta) = \varphi \text{ that } \delta \text{ dove } \varphi \in P_TV/t, \delta \in P_t$$

$$T_4(\varphi', \delta') = \varphi'(\wedge \delta')$$

Esempio: $F_4(\text{believe}, \text{Mary walks}) = \text{"believe that Mary walks"}$

$$T_4(\text{believe}', \text{walk}'(m')) = \text{believe}'(\wedge \text{walk}'(m'))$$

Regola S14: Quantifier Raising

$$F_{14}(n, \varphi, \alpha) = \varphi[\alpha/he_n] \text{ dove } \alpha \in PT_e$$

$$T_{14}(n, \varphi', \alpha') = \alpha'(\lambda x_n. \varphi')$$

Esempio: permette scope inverso in quantificatori

8.13.3 Meaning Postulates in PTQ

Definizione 8.13.2 (Postulati di significato):

PTQ include meaning postulates per catturare inferenze lessicali:

MP1: Sinonimia

$$\forall x[\text{man}'(x) \leftrightarrow \text{male-adult-human}'(x)]$$

MP2: Iperonimia

$$\forall x[\text{bachelor}'(x) \rightarrow \text{man}'(x)]$$

MP3: Relazioni temporali

$$\forall x, t[\text{walk-in-past}'(x, t) \leftrightarrow \exists t' [t' < t \wedge \text{walk}'(x, t')]]$$

MP4: Atteggiamenti proposizionali

$$\forall x, p[\text{believe}'(x, p) \rightarrow \neg \text{know}'(x, \neg p)]$$

8.13.4 Intensionalità Sistemica

Teorema 8.13.1 (Principio di intensionalità in PTQ):

Per ogni categoria C , esiste C^s (versione intensionale):

$$[[\cdot]]_C: C \rightarrow D_T$$

$$[[\cdot]]_{C^s}: C \rightarrow D_{\langle s, T \rangle}$$

Regola generale:

$$[[\alpha]]^s = \wedge [[\alpha]]$$

Esempio 8.13.1:

temperatura: e

$\wedge$ temperatura: $\langle s, e \rangle$ (funzione da indici a numeri)

"La temperatura sale" richiede versione intensionale:

$\llbracket \text{sale} \rrbracket (^{\text{temperatura}})$

8.13.5 Generalized Quantifiers in PTQ - Teoria Completa

Definizione 8.13.3 (GQ type-raised terms):

Tutti i termini in PTQ hanno tipo:

$\langle \langle e, t \rangle, t \rangle$

Interpretazione set-teorica:

$\llbracket \text{DET CN} \rrbracket = \lambda P. Q(\llbracket \text{CN} \rrbracket)(P)$

dove Q è il quantificatore generalizzato.

Teorema 8.13.2 (Living on property):

Per ogni GQ in linguaggio naturale:

$Q(A)(B) \Leftrightarrow Q(A)(A \cap B)$ (Conservativity)

Dimostrazione estesa:

Consideriamo "most":

$\llbracket \text{most} \rrbracket (A)(B) = |A \cap B| > |A - B|$

Verifichiamo conservatività:

$|A \cap B| > |A - B|$

$\Leftrightarrow |A \cap B| > |(A - B) - B|$ (B e A-B disgiunti rispetto a B)

$\Leftrightarrow |A \cap B| > |A - (A \cap B)|$

$\Leftrightarrow \llbracket \text{most} \rrbracket (A)(A \cap B)$

I

8.13.6 Intersective vs Subsective vs Non-subsective Modifiers**Definizione 8.13.4 (Tassonomia degli aggettivi):****Intersective:**

$$\llbracket A \ N \rrbracket = \llbracket A \rrbracket \cap \llbracket N \rrbracket$$

Esempio: "red car" = red things $\cap$ cars**Subsective:**

$$\llbracket A \ N \rrbracket \subseteq \llbracket N \rrbracket \text{ ma non necessariamente } \llbracket A \rrbracket \cap \llbracket N \rrbracket$$

Esempio: "large mouse" $\subseteq$ mice, ma non large things in assoluto**Non-subsective:**

$$\llbracket A \ N \rrbracket \not\subseteq \llbracket N \rrbracket$$

Esempio: "former president" $\not\subseteq$ presidents**Teorema 8.13.3 (Type assignment):**Intersective: $\llbracket A \rrbracket: \langle e, t \rangle$ Subsective: $\llbracket A \rrbracket: \langle \langle e, t \rangle, \langle e, t \rangle \rangle$ Non-subsective: $\llbracket A \rrbracket: \langle \langle e, t \rangle, \langle e, t \rangle \rangle$ (stesso tipo, semantica diversa)**8.13.7 Coordinazione e Type Polymorphism****Definizione 8.13.5 (Schema di coordinazione):**

$$\llbracket \alpha \text{ AND } \beta \rrbracket_{\tau} = \lambda x_{\tau}. \llbracket \alpha \rrbracket(x) \wedge \llbracket \beta \rrbracket(x)$$

per ogni tipo τ .**Esempio 8.13.2:** Diversi tipi coordinati

(t) "John walks and Mary runs"

$$= \text{walk}'(j) \wedge \text{run}'(m)$$

(<e,t>) "red and large"

$$= \lambda x. \text{red}'(x) \wedge \text{large}'(x)$$

(<<e,t>,t>) "every man and some woman"

$$= \lambda P. \text{every-man}(P) \wedge \text{some-woman}(P)$$

Constraint: α e β devono avere **stesso tipo**.

8.13.8 Scope Islands e Constraints

Definizione 8.13.6 (Island Constraints in LF):

Certi domini sintattici bloccano QR (Quantifier Raising):

Complex NP Island:

$$*[\text{QP}_1 \dots [\text{NP} \dots t_1 \dots] \dots]$$

Esempio 8.13.3:

X "Every paper_i, I met [NP the author of t_i]"

(quantifier non può uscire da NP complesso)

Coordinate Structure Constraint:

$$*[\text{QP}_1 \dots [\text{CoordP } \alpha \text{ and } [\dots t_1 \dots]] \dots]$$

Teorema 8.13.4 (Scope Ambiguity Restrictions):

Per n quantificatori $Q_1, \dots, Q_n$:

- Senza islands: n! letture possibili
- Con islands: $\leq n!$ letture (molte bloccate)

8.13.9 Montague's Universal Grammar Hypothesis

Congettura 8.13.1 (UG di Montague):

Ogni linguaggio naturale ha una grammatica della forma:

$$G = \langle \text{Syn}, \text{Sem}, F \rangle$$

dove:

1. Syn: algebra sintattiche libere su categorie
2. Sem: algebre semantiche (tipi in IL)
3. F: omomorfismo da Syn a Sem

Universalità:

Le categorie semantiche (tipi) sono **universali**:

$e, t, \langle e, t \rangle, \langle \langle e, t \rangle, t \rangle, \langle s, t \rangle, \dots$

La variazione inter-linguistica è solo in:

- Lessico
- Regole sintattiche F
- Parametri morfologici

8.13.10 Critiche e Limitazioni di PTQ

Problema 1: Overgeneration

PTQ genera frasi agrammaticali:

*"John every man" (type-correct ma ungrammaticale)

Problema 2: Type explosion

Troppi tipi diversi per simili categorie:

TV: $\langle e, \langle e, t \rangle \rangle$

TV/t: $\langle \langle s, t \rangle, \langle e, t \rangle \rangle$

TV/e: $\langle e, \langle e, t \rangle \rangle$

...

Problema 3: Non-compositionality

Idioms, collocazioni non trattabili:

"kick the bucket" $\neq \lambda x. \text{kick}'(x, \text{the-bucket})$

Problema 4: Discourse

PTQ è puramente frasale, non discorsuale (risolto da DRT/DPL).

8.13.11 Moderne Evoluzioni

1. Categorical Grammar (Steedman, Moortgat)

Più flessibile sintatticamente, composizionalità preservata.

2. HPSG / LFG

Constraint-based, non derivazionale.

3. Minimalist Semantics

Interfaccia con sintassi minimalista.

4. Formal Semantics in CCG

Combina flessibilità categoriale con semantica Montague-style.

8.14 Conclusioni e Prospettive Future

8.14.1 Eredità di Montague

Teorema fondamentale (informale):

Montague ha dimostrato che:

Il linguaggio naturale è un linguaggio formale

nel senso che ammette:

1. Sintassi compositiva rigorosa
2. Semantica model-teoretica
3. Interfaccia sintassi-semantica sistematica

8.14.2 Programma di Ricerca Montague

Domande ancora aperte:

1. **Quantificazione complessa:** Come trattare tutti i fenomeni di scope?
2. **Pragmatica:** Interfaccia semantica-pragmatica?
3. **Vaghezza:** Come modellare predicati vaghi?
4. **Contesto:** Dipendenza contestuale oltre intensioni?
5. **Multimodalità:** Semantica di gesti, prosodia?

8.14.3 Direzioni Contemporanee

1. Compositional Distributional Semantics

Combina Montague con vector spaces:

$[[\varphi]]$: sintassi $\rightarrow$ vector semantico

2. Neural Compositional Semantics

Composizionalità in reti neurali.

3. Quantum Semantics

Uso di spazi di Hilbert per modellare significato.

4. Game-Theoretical Semantics

Significato come strategia in giochi.

8.14.4 Montague nel XXI Secolo

La semantica di Montague rimane:

- **Fondamentale** per tutta semantica formale
- **Metodologicamente rigorosa**
- **Empiricamente testabile**
- **Computazionalmente implementabile**

Il suo lascito: l'idea che la matematica è lo strumento giusto per capire il linguaggio.

Parte VII: Fondamenti Matematici Avanzati e Teoria della Complessità

Capitolo 9: Logiche Temporali e Modalità

9.1 Sistemi di Logica Temporale Proposizionale

La logica temporale fornisce un formalismo rigoroso per il trattamento degli aspetti temporali del linguaggio naturale, estendendo la logica proposizionale classica con operatori che quantificano su istanti temporali.

Definizione 9.1.1 (Struttura temporale): Una struttura temporale (o frame temporale) è una coppia $F = \langle T, < \rangle$ dove:

(i) T è un insieme non vuoto di istanti temporali (ii) $<$ è una relazione binaria su T (la relazione di precedenza temporale)

La relazione $<$ deve soddisfare proprietà appropriate a seconda del tipo di tempo considerato.

Definizione 9.1.2 (Tipi di strutture temporali):

(i) **Tempo lineare**: $<$ è un ordine totale (transitiva, antisimmetrica, totale) $\forall t_1, t_2, t_3 \in T: [(t_1 < t_2 \wedge t_2 < t_3) \rightarrow t_1 < t_3]$ (transitività) $\forall t_1, t_2 \in T: [(t_1 < t_2 \wedge t_2 < t_1) \rightarrow t_1 = t_2]$ (antisimmetria) $\forall t_1, t_2 \in T: [t_1 < t_2 \vee t_1 = t_2 \vee t_2 < t_1]$ (totalità)

(ii) **Tempo ramificato**: $<$ è un ordine parziale con proprietà di ramificazione: Per ogni $t \in T$, l'insieme $\{s \in T : s < t\}$ è totalmente ordinato (il passato è lineare) Possono esistere t_1, t_2 con un predecessore comune ma $t_1 \not< t_2$ e $t_2 \not< t_1$ (il futuro ramifica)

(iii) **Tempo denso**: $<$ è denso, ovvero: $\forall t_1, t_2 \in T: [t_1 < t_2 \rightarrow \exists t_3 \in T: t_1 < t_3 < t_2]$

(iv) **Tempo discreto**: Ogni elemento non massimale ha un successore immediato: $\forall t \in T: [\neg \text{massimale}(t) \rightarrow \exists s \in T: (t < s \wedge \neg \exists r: t < r < s)]$

Definizione 9.1.3 (Linguaggio della logica temporale proposizionale): Il linguaggio LTL (Linear Temporal Logic) è definito dalla seguente grammatica BNF:

$\varphi ::= p \mid \neg \varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid X\varphi \mid F\varphi \mid G\varphi \mid (\varphi U \varphi) \mid (\varphi R \varphi)$

dove:

- $p \in \text{Prop}$ (insieme numerabile di proposizioni atomiche)
- X (neXt): "nel prossimo istante"
- F (Future/Eventually): "ad un certo punto nel futuro"
- G (Globally/Always): "sempre nel futuro"
- U (Until): "fino a quando"
- R (Release): "rilasciato da"

Definizione 9.1.4 (Semantica di Kripke per LTL): Un modello temporale è una tripla $M = \langle T, <, V \rangle$ dove:

- $\langle T, < \rangle$ è una struttura temporale

- $V : \text{Prop} \rightarrow \mathcal{P}(T)$ è una funzione di valutazione che assegna ad ogni proposizione atomica l'insieme degli istanti in cui è vera

Per un istante $t \in T$, la relazione di soddisfacimento $M, t \models \varphi$ è definita induttivamente:

- (i) $M, t \models p$ sse $t \in V(p)$
- (ii) $M, t \models \neg\varphi$ sse $M, t \not\models \varphi$
- (iii) $M, t \models \varphi \wedge \psi$ sse $M, t \models \varphi$ e $M, t \models \psi$
- (iv) $M, t \models X\varphi$ sse esiste $s \in T$ tale che $t < s$, non esiste r con $t < r < s$, e $M, s \models \varphi$ (nel tempo discreto; per il tempo denso si usa una definizione diversa)
- (v) $M, t \models F\varphi$ sse esiste $s \in T$ con $t < s$ tale che $M, s \models \varphi$
- (vi) $M, t \models G\varphi$ sse per ogni $s \in T$ con $t < s$: $M, s \models \varphi$
- (vii) $M, t \models \varphi U \psi$ sse esiste $s \in T$ con $t < s$ tale che $M, s \models \psi$ e per ogni r con $t < r < s$: $M, r \models \varphi$
- (viii) $M, t \models \varphi R \psi$ sse per ogni s con $t < s$: [$M, s \models \psi$ oppure esiste r con $t \leq r < s$ tale che $M, r \models \varphi$]

Teorema 9.1.1 (Interdefinibilità degli operatori temporali): Gli operatori temporali sono interdefinibili:

- (i) $F\varphi \equiv \top U \varphi$ (dove $\top$ è la tautologia) (ii) $G\varphi \equiv \neg F\neg\varphi$ (iii) $\varphi R \psi \equiv \neg(\neg\varphi U \neg\psi)$ (iv) $\varphi U \psi \equiv \neg(\neg\varphi R \neg\psi)$

Dimostrazione: Dimostriamo (ii) per illustrare la tecnica.

($\Rightarrow$) Supponiamo $M, t \models G\varphi$. Allora per ogni $s > t$, $M, s \models \varphi$. Quindi non esiste $s > t$ tale che $M, s \models \neg\varphi$, ovvero $M, t \not\models F\neg\varphi$, quindi $M, t \models \neg F\neg\varphi$.

($\Leftarrow$) Supponiamo $M, t \models \neg F\neg\varphi$. Allora $M, t \not\models F\neg\varphi$, quindi non esiste $s > t$ con $M, s \models \neg\varphi$. Pertanto per ogni $s > t$, $M, s \models \varphi$, ovvero $M, t \models G\varphi$. ■

Definizione 9.1.5 (Validità e conseguenza logica):

- (i) Una formula φ è valida in un frame F ($F \models \varphi$) se per ogni valutazione V e ogni $t \in T$: $\langle F, V \rangle, t \models \varphi$
- (ii) Una formula φ è valida ($\models \varphi$) se è valida in ogni frame
- (iii) φ è conseguenza logica di Γ ($\Gamma \models \varphi$) se per ogni modello M e istante t : se $M, t \models \psi$ per ogni $\psi \in \Gamma$, allora $M, t \models \varphi$

Teorema 9.1.2 (Assiomi di LTL): Il seguente sistema assiomatico è sound e complete per LTL su tempo lineare futuro:

Assiomi proposizionali classici più:

(T1) $G(\varphi \rightarrow \psi) \rightarrow (G\varphi \rightarrow G\psi)$ (distributività di G) (T2) $G\varphi \rightarrow \varphi$ (riflessività) (T3) $G\varphi \rightarrow GG\varphi$ (transitività) (T4) $F\varphi \leftrightarrow \varphi \vee XF\varphi$ (unfold di F) (T5) $G\varphi \leftrightarrow \varphi \wedge XG\varphi$ (unfold di G) (T6) $\varphi U\psi \leftrightarrow \varphi \vee (\varphi \wedge X(\varphi U\psi))$ (unfold di U) (T7) $G(\varphi \rightarrow X\varphi) \rightarrow (\varphi \rightarrow G\varphi)$ (induzione)

Regole di inferenza:

- Modus Ponens: da φ e $\varphi \rightarrow \psi$ inferire ψ
- Necessitazione: da φ inferire $G\varphi$

Dimostrazione (Schema della completezza): La dimostrazione di completezza procede costruendo un modello canonico a partire da insiemi massimalmente consistenti di formule. Dato un insieme Γ di formule consistente, si costruisce il modello M_Γ dove:

- $T_\Gamma = \{\Delta : \Delta \text{ è un insieme massimalmente consistente contenente tutte le "X-successioni" di } \Gamma\}$
- $\Delta_1 < \Delta_2$ sse tutte le formule $X\varphi \in \Delta_2$ soddisfano $\varphi \in \Delta_1$
- $V(p) = \{\Delta \in T_\Gamma : p \in \Delta\}$

Si dimostra per induzione sulla complessità di φ che:

$M_\Gamma, \Delta \models \varphi$ sse $\varphi \in \Delta$

Quindi se $\Gamma \models \varphi$, allora $\Gamma \vdash \varphi$. ■

9.2 Logica Temporale del Primo Ordine

Definizione 9.2.1 (FOTL - First-Order Temporal Logic): Il linguaggio FOTL estende la logica del primo ordine con operatori temporali:

$\varphi ::= R(t_1, \dots, t_n) \mid t_1 = t_2 \mid \neg\varphi \mid (\varphi \wedge \psi) \mid \exists x \varphi \mid X\varphi \mid F\varphi \mid G\varphi \mid (\varphi U \psi)$

dove R sono simboli relazionali e t sono termini (variabili o costanti).

Definizione 9.2.2 (Semantica di FOTL): Un modello è una struttura $M = \langle T, <, \{D_t\}_{t \in T}, \{I_t\}_{t \in T} \rangle$ dove:

(i) $\langle T, < \rangle$ è una struttura temporale (ii) D_t è il dominio degli individui all'istante t (iii) I_t interpreta i simboli relazionali e le costanti all'istante t

La soddisfacibilità $M, t, g \models \varphi$ (dove g è un assignment di variabili) è definita con le clausole standard per la logica del primo ordine più le clausole temporali della Definizione 9.1.4.

Teorema 9.2.1 (Indecidibilità di FOTL): La validità in FOTL è indecidibile, anche per frammenti relativamente semplici.

Dimostrazione (Schema): Si riduce il problema della fermata delle macchine di Turing a FOTL. Data una macchina di Turing M e input w, si costruisce una formula $\varphi_{\{M,w\}}$ tale che $\varphi_{\{M,w\}}$ è soddisfacibile se e solo se M si ferma su w. La formula codifica la computazione di M come una sequenza temporale di configurazioni. ■

Definizione 9.2.3 (Domini variabili vs costanti):

(i) **Domini costanti:** $D_t = D$ per ogni $t \in T$ (tutti gli individui esistono in ogni istante) (ii) **Domini variabili:** D_t può variare con t (individui possono venire ad esistenza e cessare di esistere)

Con domini variabili, distinguiamo quantificatori:

- $\exists x \varphi$: esiste un individuo nel dominio corrente
- $\exists_A x \varphi$: esiste attualmente un individuo
- $\exists_E x \varphi$: esisterà eventualmente un individuo

Teorema 9.2.2 (Barcan Formula): Con domini costanti, vale la Barcan Formula:

$$\forall x G\varphi \rightarrow G\forall x \varphi$$

e la sua converso. Con domini variabili, queste formule non sono generalmente valide.

Dimostrazione: Con domini costanti, ogni individuo esiste in ogni istante. Quindi:

($\Rightarrow$) Se $M, t \models \forall x G\varphi$, allora per ogni $d \in D$ e ogni $s > t$: $M, s, g[x \mapsto d] \models \varphi$. Ma poiché $D_s = D$, per ogni $s > t$: per ogni $d \in D_s$: $M, s, g[x \mapsto d] \models \varphi$, ovvero $M, s \models \forall x \varphi$. Quindi $M, t \models G\forall x \varphi$.

Con domini variabili, se un individuo d esiste in t ma non in $s > t$, allora $\forall x G\varphi$ può essere vero (perché $G\varphi$ vale per d finché d esiste), ma $G\forall x \varphi$ può essere falso (perché in s potrebbe non esistere alcun individuo, rendendo $\forall x \varphi$ vacuamente vero in t ma non necessariamente in s). ■

9.3 Logiche Modali e Strutture di Kripke

Definizione 9.3.1 (Frame modale): Un frame modale è una coppia $F = \langle W, R \rangle$ dove:

(i) W è un insieme non vuoto di mondi possibili (ii) $R \subseteq W \times W$ è una relazione di accessibilità

Definizione 9.3.2 (Modello di Kripke): Un modello di Kripke è una tripla $M = \langle W, R, V \rangle$ dove:

(i) $\langle W, R \rangle$ è un frame modale (ii) $V : \text{Prop} \rightarrow \mathcal{P}(W)$ è una funzione di valutazione

Definizione 9.3.3 (Linguaggio della logica modale): Il linguaggio della logica modale proposizionale è:

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid \Box\varphi \mid \Diamond\varphi$$

dove $\Box$ è il necessità e $\Diamond$ è possibilità.

Definizione 9.3.4 (Semantica modale): Per un mondo $w \in W$:

(i) $M, w \models p$ sse $w \in V(p)$ (ii) $M, w \models \neg\phi$ sse $M, w \not\models \phi$ (iii) $M, w \models \phi \wedge \psi$ sse $M, w \models \phi$ e $M, w \models \psi$ (iv) $M, w \models \Box\phi$ sse per ogni v con wRv : $M, v \models \phi$ (v) $M, w \models \Diamond\phi$ sse esiste v con wRv tale che $M, v \models \phi$

Teorema 9.3.1 (Dualità): $\Diamond\phi \equiv \neg\Box\neg\phi$ e $\Box\phi \equiv \neg\Diamond\neg\phi$

Definizione 9.3.5 (Proprietà della relazione di accessibilità e corrispondenti assiomi):

(i) **Riflessività**: $\forall w \in W: wRw$ Corrispondente assioma T: $\Box\phi \rightarrow \phi$

(ii) **Transitività**: $\forall u, v, w \in W: (uRv \wedge vRw) \rightarrow uRw$ Corrispondente assioma 4: $\Box\phi \rightarrow \Box\Box\phi$

(iii) **Simmetria**: $\forall u, v \in W: uRv \rightarrow vRu$ Corrispondente assioma B: $\phi \rightarrow \Box\Diamond\phi$

(iv) **Euclidea**: $\forall u, v, w \in W: (uRv \wedge uRw) \rightarrow vRw$ Corrispondente assioma 5: $\Diamond\phi \rightarrow \Box\Diamond\phi$

(v) **Serialità**: $\forall w \in W: \exists v: wRv$ Corrispondente assioma D: $\Box\phi \rightarrow \Diamond\phi$

Teorema 9.3.2 (Corrispondenza): Per ogni proprietà P della relazione di accessibilità sopra menzionata, una formula ϕ è valida in tutti i frame che soddisfano P se e solo se ϕ è derivabile nel sistema modale corrispondente.

Dimostrazione (per riflessività e assioma T):

($\Rightarrow$) Supponiamo che R sia riflessiva. Vogliamo mostrare che $\Box\phi \rightarrow \phi$ è valida. Sia M un modello su un frame riflessivo, e sia $w \in W$. Se $M, w \models \Box\phi$, allora per ogni v con wRv : $M, v \models \phi$. Poiché R è riflessiva, wRw , quindi $M, w \models \phi$. Dunque $M, w \models \Box\phi \rightarrow \phi$.

($\Leftarrow$) Supponiamo che $\Box\phi \rightarrow \phi$ sia valida in un frame $F = \langle W, R \rangle$. Vogliamo mostrare che R è riflessiva. Per contraddizione, supponiamo che esista $w \in W$ tale che $\neg(wRw)$. Definiamo una valutazione $V(p) = W \setminus \{w\}$. Allora per ogni v con wRv (ce ne sono alcuni, per serialità, ma non w), $v \in V(p)$, quindi $M, v \models p$. Dunque $M, w \models \Box p$. Ma $M, w \not\models p$ poiché $w \notin V(p)$. Quindi $M, w \not\models \Box p \rightarrow p$, contraddicendo la validità di T. ■

Definizione 9.3.6 (Sistemi modali standard):

- **K**: sistema modale minimale (solo distributività $\Box(\phi \rightarrow \psi) \rightarrow (\Box\phi \rightarrow \Box\psi)$)
- **T**: K + assioma T ($\Box\phi \rightarrow \phi$)
- **S4**: T + assioma 4 ($\Box\phi \rightarrow \Box\Box\phi$)
- **S5**: S4 + assioma B o 5 (relazione di equivalenza)
- **D**: K + assioma D ($\Box\phi \rightarrow \Diamond\phi$)

Teorema 9.3.3 (Completezza dei sistemi modali): Ciascuno dei sistemi K, T, S4, S5, D è sound e complete rispetto alla corrispondente classe di frame.

Teorema 9.3.4 (Decidibilità di S5): La validità in S5 è decidibile.

Dimostrazione (Schema): S5 corrisponde a frame con relazioni di equivalenza. Su tali frame, ogni formula è equivalente a una formula in forma normale modale dove gli operatori

modali non sono annidati. Il numero di tali formule è finito a meno di equivalenza logica, quindi esiste un algoritmo di decisione. ■

9.4 Bisimulazione e Invarianza Modale

Definizione 9.4.1 (Bisimulazione): Siano $M_1 = \langle W_1, R_1, V_1 \rangle$ e $M_2 = \langle W_2, R_2, V_2 \rangle$ due modelli. Una bisimulazione tra M_1 e M_2 è una relazione $Z \subseteq W_1 \times W_2$ tale che per ogni $(w_1, w_2) \in Z$:

- (i) **Atomic harmony**: Per ogni $p \in \text{Prop}$: $w_1 \in V_1(p)$ sse $w_2 \in V_2(p)$
- (ii) **Zig**: Per ogni w'_1 con $w_1 R_1 w'_1$, esiste w'_2 con $w_2 R_2 w'_2$ e $(w'_1, w'_2) \in Z$
- (iii) **Zag**: Per ogni w'_2 con $w_2 R_2 w'_2$, esiste w'_1 con $w_1 R_1 w'_1$ e $(w'_1, w'_2) \in Z$

Scriviamo $w_1 \leftrightarrow_Z w_2$ quando $(w_1, w_2) \in Z$, e diciamo che w_1 e w_2 sono bisimili.

Teorema 9.4.1 (Invarianza modale): Se $w_1 \leftrightarrow_Z w_2$, allora per ogni formula modale φ :

$M_1, w_1 \models \varphi$ sse $M_2, w_2 \models \varphi$

Dimostrazione: Per induzione sulla complessità di φ .

Base (p atomica): Immediato dalla condizione (i) di atomic harmony.

Passo induttivo (caso $\Box\varphi$):

($\Rightarrow$) Supponiamo $M_1, w_1 \models \Box\varphi$ e $w_1 \leftrightarrow_Z w_2$. Vogliamo mostrare $M_2, w_2 \models \Box\varphi$. Sia w'_2 tale che $w_2 R_2 w'_2$. Per la condizione zag, esiste w'_1 con $w_1 R_1 w'_1$ e $w'_1 \leftrightarrow_Z w'_2$. Poiché $M_1, w_1 \models \Box\varphi$, abbiamo $M_1, w'_1 \models \varphi$. Per ipotesi induttiva, $M_2, w'_2 \models \varphi$. Poiché w'_2 era arbitrario con $w_2 R_2 w'_2$, concludiamo $M_2, w_2 \models \Box\varphi$.

($\Leftarrow$) Simmetrico usando la condizione zig. ■

Teorema 9.4.2 (Caratterizzazione della bisimulazione): Due stati w_1 e w_2 sono bisimili se e solo se soddisfano esattamente le stesse formule modali.

Corollario 9.4.3: La logica modale è invariante per bisimulazione: cattura esattamente le proprietà preservate dalle bisimulazioni.

Definizione 9.4.2 (Relazione di simulazione): Una simulazione da M_1 a M_2 è una relazione $Z \subseteq W_1 \times W_2$ che soddisfa le condizioni (i) e (ii) della bisimulazione (atomic harmony e zig), ma non necessariamente zag.

Teorema 9.4.4: Se w_1 è simulato da w_2 , allora ogni formula $\Box$ -positiva (contenente solo $\Box$ e non $\Diamond$) vera in w_1 è vera anche in w_2 .

Capitolo 10: Teoria della Complessità per Linguaggi Formali

10.1 Classi di Complessità Temporale

Definizione 10.1.1 (Complessità temporale): Sia M una macchina di Turing deterministica. Il tempo di esecuzione di M su input w , denotato $\text{time}_M(w)$, è il numero di passi che M compie prima di fermarsi su input w (oppure ∞ se M non si ferma).

La funzione di complessità temporale di M è:

$$T_M(n) = \max\{\text{time}_M(w) : |w| = n\}$$

Definizione 10.1.2 (Classe DTIME): Per una funzione $t : \mathbb{N} \rightarrow \mathbb{N}$:

$$\text{DTIME}(t(n)) = \{L : \text{esiste una DTM } M \text{ che decide } L \text{ in tempo } O(t(n))\}$$

Definizione 10.1.3 (Classi di complessità standard):

- (i) $P = \bigcup_{k \geq 1} \text{DTIME}(n^k)$ (tempo polinomiale deterministico)
- (ii) $\text{EXPTIME} = \bigcup_{k \geq 1} \text{DTIME}(2^{n^k})$ (tempo esponenziale)
- (iii) $\text{NTIME}(t(n)) = \{L : \text{esiste una NTM } M \text{ che decide } L \text{ in tempo } O(t(n))\}$
- (iv) $\text{NP} = \bigcup_{k \geq 1} \text{NTIME}(n^k)$ (tempo polinomiale non deterministico)

Teorema 10.1.1 (Gerarchia temporale deterministica): Se f, g sono funzioni tempo-costruibili con $f(n) \log f(n) = o(g(n))$, allora:

$$\text{DTIME}(f(n)) \subsetneq \text{DTIME}(g(n))$$

Dimostrazione (Schema - diagonalizzazione): Costruiamo una macchina diagonale D che su input $\langle M, w \rangle$ (dove M è una DTM):

1. Simula M su w per al massimo $g(|\langle M, w \rangle|)$ passi
2. Se M accetta entro questo tempo, D rigetta; altrimenti D accetta

Si dimostra che:

- D decide un linguaggio in $\text{DTIME}(g(n))$
- Per ogni M che opera in tempo $f(n)$, esiste un input su cui D e M differiscono
- Quindi $L(D) \notin \text{DTIME}(f(n))$

Il fattore $\log f(n)$ è necessario per la simulazione overhead. ■

Teorema 10.1.2: $P \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME}$

Inoltre, $P \subsetneq \text{EXPTIME}$ (le inclusioni estreme sono proprie).

Dimostrazione ($P \subseteq \text{NP}$): Ogni DTM è un caso particolare di NTM.

($\text{NP} \subseteq \text{PSPACE}$): Una NTM che opera in tempo polinomiale usa spazio polinomiale (ogni cella visitata in tempo t è in posizione $\leq t$).

(PSPACE $\subseteq$ EXPTIME): Una DTM può esplorare sistematicamente tutte le configurazioni di una macchina che usa spazio $s(n)$, e ci sono al massimo $O(2^{cs(n)})$ configurazioni per qualche costante c . ■

10.2 Complessità dei Problemi per Linguaggi Formali

Teorema 10.2.1 (Complessità del problema della membership):

(i) Per linguaggi regolari: $O(n)$ tempo, $O(1)$ spazio (ii) Per linguaggi context-free: $O(n^3)$ tempo (algoritmo CYK), $O(n^2)$ spazio (iii) Per linguaggi context-sensitive: PSPACE-completo (iv) Per linguaggi ricorsivamente enumerabili: indecidibile in generale

Dimostrazione:

(i) Un DFA ha un numero costante di stati e fa esattamente una transizione per simbolo di input, quindi $O(n)$ tempo.

(ii) L'algoritmo CYK riempie una tabella $n \times n$, e per ogni cella (i,j) esamina $O(n)$ decomposizioni, ciascuna richiedendo $O(|P|)$ tempo. Quindi $O(n^3 \cdot |P|) = O(n^3)$.

(iii) La membership per CSL si riduce al problema di accessibilità in un grafo di configurazioni di dimensione esponenziale, risolvibile in PSPACE. Si dimostra PSPACE-hardness mediante riduzione da TQBF. ■

Definizione 10.2.1 (Linguaggio context-sensitive): Un linguaggio L è context-sensitive se esiste una macchina di Turing linearmente limitata (LBA) che lo decide, ovvero una NTM che usa al massimo $O(n)$ spazio sul nastro.

Teorema 10.2.2: La classe dei linguaggi context-sensitive coincide con NSPACE(n).

Teorema 10.2.3 (Teorema di Immerman-Szelepcsényi): NSPACE($s(n)$) è chiuso sotto complemento per $s(n) \geq \log n$.

Corollario 10.2.4: I linguaggi context-sensitive sono chiusi sotto complemento.

Definizione 10.2.2 (Complessità dell'analisi): Data una CFG G e una stringa w , il problema dell'analisi chiede di costruire un albero di derivazione per w (se esiste).

Teorema 10.2.5 (Complessità ottimale dell'analisi CF):

(i) Per CFG ambigue: $\Omega(n^2)$ nel caso peggiore (numero di alberi di derivazione può essere esponenziale) (ii) Per CFG non ambigue: $O(n)$ è possibile con preprocessing (iii) Per grammatiche LR(k): $O(n)$ tempo lineare con automa a pila deterministico

Teorema 10.2.6: Il problema di decidere se una CFG è ambigua è indecidibile.

Dimostrazione (Schema): Riduzione dal Post Correspondence Problem. Data un'istanza del PCP con liste $(u_1, \dots, u_k)$ e $(v_1, \dots, v_k)$, si costruisce una CFG G tale che G è ambigua sse il PCP ha soluzione. La grammatica ha produzioni che possono generare la stessa stringa in due modi diversi, corrispondenti alle due liste del PCP. ■

10.3 Automi con Complessità Limitata

Definizione 10.3.1 (Automa a contatore): Un automa a contatore è un PDA in cui l'alfabeto della pila contiene solo due simboli: Z_0 (simbolo di fondo) e X (simbolo contatore). Le operazioni permesse sono:

- Push(X): incrementa il contatore
- Pop(X): decrementa il contatore (se > 0)
- Test zero: verifica se il contatore è zero

Teorema 10.3.1: Gli automi a un contatore riconoscono una classe di linguaggi propriamente contenuta nei CFL.

Esempio 10.3.1: Il linguaggio $L = \{a^n b^n c^m : n, m \geq 0\}$ è riconosciuto da un automa a un contatore ma $L' = \{a^n b^m c^{n+m} : n, m \geq 0\}$ richiede due contatori e non è context-free.

Definizione 10.3.2 (Automa a pila annidato - Nested Stack Automaton): Un NSA è un automa dove la pila può contenere altre pile ricorsivamente. Ad ogni livello si possono fare operazioni push/pop, e si può "entrare" in una pila nested.

Teorema 10.3.2: Gli NSA con annidamento limitato a profondità k riconoscono linguaggi in $\text{DTIME}(n^{2^k})$.

Definizione 10.3.3 (Tree Automata): Un tree automaton è una quintupla $A = (Q, \Sigma, \delta, F, q_0)$ dove:

- Q è l'insieme finito degli stati
- Σ è un alfabeto ranked (ogni simbolo ha un'arietà)
- $\delta : \bigcup_{n \geq 0} Q^n \times \Sigma_n \rightarrow Q$ è la funzione di transizione
- $F \subseteq Q$ sono gli stati accettanti
- q_0 è lo stato iniziale

Teorema 10.3.3: Per tree automata bottom-up deterministici:

- Membership: $O(n)$ dove n è il numero di nodi dell'albero
- Emptiness: $O(|Q| \cdot |\delta|)$
- Equivalence: decidibile in tempo polinomiale

Dimostrazione (Membership): Si procede bottom-up assegnando stati alle foglie, poi ai nodi interni basandosi sugli stati dei figli. Ogni nodo è visitato esattamente una volta, quindi $O(n)$.

I

10.4 Complessità Descrittiva

Definizione 10.4.1 (Complessità descrittiva): La complessità descrittiva studia la corrispondenza tra classi di complessità e frammenti logici.

Teorema 10.4.1 (Teorema di Fagin): NP è esattamente la classe dei linguaggi definibili in logica esistenziale del secondo ordine (ESO):

$NP = ESO = \{L : L \text{ è definibile da una formula } \exists R_1 \dots \exists R_k \varphi \text{ dove } \varphi \text{ è FO}\}$

Dimostrazione (Schema):

($\Rightarrow$) Data una NTM M in tempo polinomiale $p(n)$, codifichiamo le computazioni accettanti come relazioni del secondo ordine:

- Variabili del secondo ordine per la configurazione iniziale, le transizioni, gli stati
- Formula FO φ che verifica che la computazione è valida e accettante
- La formula $\exists \text{ Config } \exists \text{ Trans } \varphi$ caratterizza esattamente $L(M)$

($\Leftarrow$) Data una formula ESO $\exists R_1 \dots \exists R_k \varphi$, costruiamo una NTM che:

- Indovina interpretazioni per $R_1, \dots, R_k$
- Verifica deterministicamente che φ è soddisfatta
- Il guess è polinomiale nella dimensione della struttura $\blacksquare$

Teorema 10.4.2 (Teorema di Immerman-Vardi): P è esattamente la classe dei linguaggi definibili in logica del primo ordine con operatore di punto fisso (FO+LFP):

$$P = FO(LFP)$$

Definizione 10.4.2 (Logica modale μ -calcolo): Il μ -calcolo estende la logica modale con operatori di punto fisso minimo (μ) e massimo (ν):

$$\varphi ::= p \mid X \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

dove X è una variabile proposizionale del secondo ordine.

Semantica: Dato un modello $M = \langle W, R, V \rangle$:

$$\llbracket \mu X.\varphi \rrbracket = \bigcap \{S \subseteq W : \llbracket \varphi \rrbracket^{V[X \mapsto S]} \subseteq S\} \text{ (minimo punto fisso)} \quad \llbracket \nu X.\varphi \rrbracket = \bigcup \{S \subseteq W : S \subseteq \llbracket \varphi \rrbracket^{V[X \mapsto S]}\} \text{ (massimo punto fisso)}$$

Teorema 10.4.3: Il μ -calcolo modale è equivalente all'automa a alberi alternante.

Teorema 10.4.4: Il problema del model checking per μ -calcolo è in $NP \cap co-NP$.

10.4 Complessità Parametrizzata

Definizione 10.4.1 (Fixed-Parameter Tractability): Un problema parametrizzato (L, k) è fixed-parameter tractable (FPT) se esiste un algoritmo che decide L in tempo: $f(k) \cdot \text{poly}(n)$ dove n è la dimensione dell'input e f è una funzione arbitraria solo del parametro k .

Teorema 10.4.1 (FPT per DFA minimization): Il problema di minimizzare un DFA con n stati a un DFA con k stati è FPT con parametro k .

Complessità: $O(2^{O(k \log k)} \cdot n \cdot \log n)$

Dimostrazione: Usiamo color-coding e tecniche di branching limitato. Il numero di partizioni possibili di n elementi in k parti è limitato dal numero di Stirling $S(n,k)$, e possiamo verificare ogni partizione in tempo polinomiale. ■

10.5 Complessità di Comunicazione

Definizione 10.5.1 (Communication Complexity): Alice ha input $x \in X$, Bob ha input $y \in Y$. Devono calcolare $f(x, y)$ comunicando il minor numero di bit.

La complessità di comunicazione deterministica è: $D(f) = \max_{x,y} \min\{|\text{protocol computing } f(x,y)|\}$

Teorema 10.5.1 (Lower bound per non-regolarità): Se L non è regolare, allora la complessità di comunicazione del problema di membership per L è $\Omega(\log n)$.

Dimostrazione: Se L è regolare con p stati, Alice può inviare a Bob il stato del DFA dopo aver letto la sua porzione di input ($\log p$ bit). Se L non è regolare, il numero di "stati" cresce, quindi la comunicazione deve crescere. ■

Teorema 10.5.2: Per il linguaggio $L = \{a^n b^n : n \geq 0\}$: $D(\text{membership in } L) = \Theta(\log n)$

10.6 Quantum Complexity per Linguaggi Formali

Definizione 10.6.1 (BQP - Bounded-Error Quantum Polynomial Time): $BQP = \{L : \exists \text{ algoritmo quantistico polinomiale che decide } L \text{ con errore } \leq 1/3\}$

Teorema 10.6.1: $REG \subsetneq BQP$ ma la relazione precisa tra CFL e BQP è aperta.

Teorema 10.6.2 (Vantaggio quantistico per pattern matching): Il pattern matching in stringhe può essere accelerato quadraticamente usando algoritmo di Grover.

Complessità classica: $O(nm)$ per pattern di lunghezza m in testo di lunghezza n

Complessità quantistica: $O(\sqrt{nm})$

Dimostrazione: L'algoritmo di Grover fornisce speedup quadratico per ricerca non strutturata, applicabile alla ricerca di occorrenze del pattern. ■

Capitolo 11: Teoria dei Modelli per Linguaggi Formali

11.1 Strutture e Teorie

Definizione 11.1.1 (Linguaggio del primo ordine): Un linguaggio del primo ordine L consiste di:

(i) Un insieme numerabile di variabili: $\text{Var} = \{x_0, x_1, x_2, \dots\}$ (ii) Simboli logici: $\neg, \wedge, \vee, \rightarrow, \forall, \exists, =$ (iii) Una signature $\sigma = (F, R, ar)$ dove: - F è un insieme di simboli funzionali - R è un insieme di simboli relazionali - $ar : F \cup R \rightarrow \mathbb{N}$ assegna arità a ciascun simbolo

Definizione 11.1.2 (L-struttura): Una L -struttura è $A = (A, \{f^A\}_{f \in F}, \{R^A\}_{R \in R})$ dove:

(i) A è un insieme non vuoto (universo o dominio) (ii) Per ogni $f \in F$ con $\text{ar}(f) = n$: $f^A : A^n \rightarrow A$ (iii) Per ogni $R \in R$ con $\text{ar}(R) = n$: $R^A \subseteq A^n$

Definizione 11.1.3 (Soddisfacimento): Data una struttura A , un assignment $g : \text{Var} \rightarrow A$, e una formula φ , definiamo $A, g \models \varphi$ induttivamente:

(i) $A, g \models R(t_1, \dots, t_n)$ sse $(\llbracket t_1 \rrbracket^A_g, \dots, \llbracket t_n \rrbracket^A_g) \in R^A$

(ii) $A, g \models t_1 = t_2$ sse $\llbracket t_1 \rrbracket^A_g = \llbracket t_2 \rrbracket^A_g$

(iii) $A, g \models \neg \varphi$ sse $A, g \not\models \varphi$

(iv) $A, g \models \varphi \wedge \psi$ sse $A, g \models \varphi$ e $A, g \models \psi$

(v) $A, g \models \exists x \varphi$ sse esiste $a \in A$: $A, g[x \mapsto a] \models \varphi$

Definizione 11.1.4 (Teoria): Una teoria T è un insieme di sentenze (formule chiuse). Il modello di T è una struttura A tale che $A \models \varphi$ per ogni $\varphi \in T$.

Definizione 11.1.5 (Conseguenza logica): Una formula φ è conseguenza logica di T (scritto $T \models \varphi$) se ogni modello di T è un modello di φ .

Teorema 11.1.1 (Teorema di Completezza di Gödel): Per la logica del primo ordine:

$T \models \varphi$ sse $T \vdash \varphi$

dove $\vdash$ denota derivabilità in un sistema di deduzione completo (es. deduzione naturale, calcolo dei sequenti).

Dimostrazione (Schema):

($\Rightarrow$) Soundness: Per induzione sulla lunghezza della derivazione.

($\Leftarrow$) Completezza: Costruzione del modello canonico. Se $T \not\models \varphi$, allora $T \cup \{\neg \varphi\}$ è consistente. Si estende a un insieme massimalmente consistente Γ , e si costruisce un modello A_Γ dove:

- Universo = termini chiusi / $\approx$ (dove $t \approx s$ sse $t = s \in \Gamma$)
- $f^A_\Gamma(\llbracket t_1 \rrbracket, \dots, \llbracket t_n \rrbracket) = \llbracket f(t_1, \dots, t_n) \rrbracket$
- $(\llbracket t_1 \rrbracket, \dots, \llbracket t_n \rrbracket) \in R^A_\Gamma$ sse $R(t_1, \dots, t_n) \in \Gamma$

Si dimostra per induzione che $A_\Gamma \models \psi$ sse $\psi \in \Gamma$. Quindi $A_\Gamma \models T$ e $A_\Gamma \models \neg \varphi$, quindi $T \not\models \varphi$.
I

11.2 Teoremi Fondamentali della Teoria dei Modelli

Teorema 11.2.1 (Teorema di Compatezza): Sia T un insieme di sentenze. Se ogni sottoinsieme finito di T ha un modello, allora T ha un modello.

Dimostrazione: Equivalente alla completezza di Gödel. Se T non ha modello, allora $T \models \perp$ (contraddizione). Per completezza, $T \vdash \perp$, quindi esiste una derivazione finita, che usa solo

un sottoinsieme finito $T_0 \subseteq T$. Ma allora $T_0 \models \perp$, quindi T_0 non ha modello, contraddicendo l'ipotesi. ■

Teorema 11.2.2 (Teorema di Löwenheim-Skolem discendente): Se T ha un modello infinito, allora ha un modello numerabile.

Dimostrazione: Dato un modello A di cardinalità $\kappa \geq \aleph_0$, si costruisce un sottomodello numerabile $B \subseteq A$:

1. Parti con un insieme numerabile $B_0 \subseteq A$ (es. contenente le interpretazioni delle costanti)
2. Per ogni n , definisce B_{n+1} chiudendo B_n sotto:
 - Applicazione di funzioni
 - Testimoni per quantificatori esistenziali
3. $B = \bigcup_{n \in \mathbb{N}} B_n$ è numerabile e è un sottomodello elementare di A

Si dimostra per induzione che $B \models \varphi$ sse $A \models \varphi$ per ogni sentenza φ . ■

Teorema 11.2.3 (Teorema di Löwenheim-Skolem ascendente): Se T ha modelli infiniti arbitrariamente grandi, allora per ogni cardinale $\kappa \geq |L|$, T ha un modello di cardinalità κ .

Definizione 11.2.1 (Equivalenza elementare): Due strutture A e B sono elementarmente equivalenti ($A \equiv B$) se soddisfano le stesse sentenze:

$\forall \varphi$ sentenza: $A \models \varphi$ sse $B \models \varphi$

Definizione 11.2.2 (Isomorfismo): Un isomorfismo tra A e B è una biiezione $h : A \rightarrow B$ tale che:

- (i) Per ogni $f \in F$ di arità n e $a_1, \dots, a_n \in A$: $h(f^A(a_1, \dots, a_n)) = f^B(h(a_1), \dots, h(a_n))$
- (ii) Per ogni $R \in R$ di arità n e $a_1, \dots, a_n \in A$: $(a_1, \dots, a_n) \in R^A$ sse $(h(a_1), \dots, h(a_n)) \in R^B$

Teorema 11.2.4: Se $A \cong B$ (isomorfe), allora $A \equiv B$.

Il converso non vale in generale: strutture elementarmente equivalenti possono non essere isomorfe (es. $\mathbb{Q}$ e $\mathbb{R}$ come ordini densi).

Definizione 11.2.3 (Sottostruttura elementare): B è una sottostruttura elementare di A (scritto $B \leq A$) se $B \subseteq A$ e per ogni formula $\varphi(x_1, \dots, x_n)$ e $b_1, \dots, b_n \in B$:

$B \models \varphi(b_1, \dots, b_n)$ sse $A \models \varphi(b_1, \dots, b_n)$

Teorema 11.2.5 (Criterio di Tarski-Vaught): $B \subseteq A$ è una sottostruttura elementare sse per ogni formula $\varphi(x, y)$ e $b \in B$: se $A \models \exists x \varphi(x, b)$, allora esiste $b' \in B$ con $A \models \varphi(b', b)$.

11.3 Categoricità e Completezza

Definizione 11.3.1 (Teoria κ -categorica): Una teoria T è κ -categorica se ogni due modelli di T di cardinalità κ sono isomorfi.

Teorema 11.3.1 (Teorema di Morley): Se una teoria T numerabile in un linguaggio numerabile è κ -categorica per qualche κ non numerabile, allora è κ -categorica per ogni κ non numerabile.

Definizione 11.3.2 (Teoria completa): Una teoria T è completa se per ogni sentenza φ : $T \models \varphi$ oppure $T \models \neg\varphi$.

Teorema 11.3.2: Una teoria T è completa sse tutti i suoi modelli sono elementarmente equivalenti.

Dimostrazione:

($\Rightarrow$) Se T è completa e $A, B \models T$, allora per ogni sentenza φ : $T \models \varphi$ o $T \models \neg\varphi$. Nel primo caso $A \models \varphi$ e $B \models \varphi$, nel secondo $A \models \neg\varphi$ e $B \models \neg\varphi$. Quindi $A \equiv B$.

($\Leftarrow$) Se tutti i modelli di T sono elementarmente equivalenti, sia φ una sentenza. Se $T \not\models \varphi$, allora esiste un modello A di T con $A \not\models \varphi$, quindi $A \models \neg\varphi$. Per equivalenza elementare, ogni modello di T soddisfa $\neg\varphi$, quindi $T \models \neg\varphi$. ■

Esempio 11.3.1: La teoria DLO degli ordini densi senza estremi è completa e $\aleph_0$ -categorica (ogni modello numerabile è isomorfo a $\mathbb{Q}$).

Trattato sulla Teoria Formale delle Lingue Naturali

Parte VIII: Teoria delle Categorie e Semantica Denotazionale

Capitolo 12: Fondamenti Categoriali per la Semantica

12.1 Categorie e Funtori

La teoria delle categorie fornisce un linguaggio astratto unificante per descrivere strutture matematiche e le relazioni tra di esse, particolarmente utile nella semantica formale dei linguaggi.

Definizione 12.1.1 (Categoria): Una categoria C consiste di:

(i) Una classe $\text{Ob}(C)$ di oggetti (ii) Per ogni coppia di oggetti A, B , un insieme $C(A, B)$ (o $\text{Hom}_C(A, B)$) di morfismi (o frecce) da A a B (iii) Per ogni oggetto A , un morfismo identità $\text{id}_A \in C(A, A)$ (iv) Un'operazione di composizione $\circ : C(B, C) \times C(A, B) \rightarrow C(A, C)$ per ogni tripla A, B, C

soggetti ai seguenti assiomi:

(a) **Associatività:** Per ogni $f : A \rightarrow B$, $g : B \rightarrow C$, $h : C \rightarrow D$: $h \circ (g \circ f) = (h \circ g) \circ f$

(b) **Identità:** Per ogni $f : A \rightarrow B$: $f \circ \text{id}_A = f = \text{id}_B \circ f$

Esempio 12.1.1 (Categorie fondamentali):

(i) **Set:** oggetti = insiemi, morfismi = funzioni (ii) **Grp:** oggetti = gruppi, morfismi = omomorfismi di gruppi (iii) **Top:** oggetti = spazi topologici, morfismi = funzioni continue (iv) **Poset:** dato un insieme parzialmente ordinato $(P, \leq)$, si definisce una categoria con $\text{Ob} = P$ e esattamente un morfismo $a \rightarrow b$ sse $a \leq b$ (v) **Mon:** oggetti = monoidi, morfismi = omomorfismi di monoidi

Definizione 12.1.2 (Isomorfismo): Un morfismo $f : A \rightarrow B$ è un isomorfismo se esiste $g : B \rightarrow A$ tale che:

$$g \circ f = \text{id}_A \text{ e } f \circ g = \text{id}_B$$

Scriviamo $A \cong B$ se esiste un isomorfismo tra A e B .

Definizione 12.1.3 (Funtore): Un funtore $F : C \rightarrow D$ tra categorie C e D consiste di:

(i) Una funzione oggetti $F_{\text{obj}} : \text{Ob}(C) \rightarrow \text{Ob}(D)$ (ii) Per ogni $A, B \in \text{Ob}(C)$, una funzione morfismi $F_{\{A,B\}} : C(A,B) \rightarrow D(F(A), F(B))$

soddisfacenti:

(a) $F(\text{id}_A) = \text{id}_{F(A)}$ per ogni $A \in \text{Ob}(C)$ (b) $F(g \circ f) = F(g) \circ F(f)$ per ogni $f : A \rightarrow B$, $g : B \rightarrow C$

Spesso si omette il pedice e si scrive semplicemente F per entrambe le funzioni.

Esempio 12.1.2: Il funtore "insieme potenza" $P : \text{Set} \rightarrow \text{Set}$ definito da:

- $P(X) = \{A : A \subseteq X\}$
- $P(f : X \rightarrow Y)(A) = \{f(a) : a \in A\} = f[A]$

Definizione 12.1.4 (Funtore controvariante): Un funtore controvariante $F : C \rightarrow D$ inverte la direzione delle frecce:

$$F : C^{\text{op}} \rightarrow D$$

dove C^{op} è la categoria opposta (stessi oggetti, frecce invertite). Equivalentemente, F soddisfa:

$$F(g \circ f) = F(f) \circ F(g)$$

Esempio 12.1.3: Il funtore controvariante $\text{Hom}_C(-, B) : C^{\text{op}} \rightarrow \text{Set}$ per un oggetto fisso B , definito da:

- $\text{Hom}_C(A, B) =$ insieme dei morfismi da A a B
- Per $f : A' \rightarrow A$, $\text{Hom}_C(f, B)(h : A \rightarrow B) = h \circ f$

Definizione 12.1.5 (Trasformazione naturale): Siano $F, G : C \rightarrow D$ funtori. Una trasformazione naturale $\eta : F \Rightarrow G$ è una famiglia di morfismi $\{\eta_A : F(A) \rightarrow G(A)\}_{A \in \text{Ob}(C)}$ tale che per ogni $f : A \rightarrow B$ in C , il seguente diagramma commuta:

$$\begin{array}{ccc} F(A) & \xrightarrow{\eta_A} & G(A) \\ | & & | \\ F(f) & & G(f) \\ | & & | \\ F(B) & \xrightarrow{\eta_B} & G(B) \end{array}$$

ovvero: $G(f) \circ \eta_A = \eta_B \circ F(f)$

Teorema 12.1.1: Le categorie, i funtori e le trasformazioni naturali formano una 2-categoria **Cat**, dove:

- 0-celle = categorie
- 1-celle = funtori
- 2-celle = trasformazioni naturali

Definizione 12.1.6 (Isomorfismo naturale): Una trasformazione naturale $\eta : F \Rightarrow G$ è un isomorfismo naturale se ogni η_A è un isomorfismo.

Lemma 12.1.2 (Lemma di Yoneda): Sia C una categoria localmente piccola, $A \in \text{Ob}(C)$, e $F : C \rightarrow \text{Set}$ un funtore. Allora:

$$\text{Nat}(\text{Hom}_C(A, -), F) \cong F(A)$$

dove Nat denota l'insieme delle trasformazioni naturali.

L'isomorfismo è dato da $\eta \mapsto \eta_A(\text{id}_A)$.

Dimostrazione: Definiamo $\Phi : \text{Nat}(\text{Hom}_C(A, -), F) \rightarrow F(A)$ da $\Phi(\eta) = \eta_A(\text{id}_A)$.

Definiamo $\Psi : F(A) \rightarrow \text{Nat}(\text{Hom}_C(A, -), F)$ come segue: dato $x \in F(A)$, costruiamo η^x tale che per ogni B e $f : A \rightarrow B$:

$$\eta^x_B(f) = F(f)(x)$$

Verifichiamo che η^x è una trasformazione naturale. Per $g : B \rightarrow C$ e $f : A \rightarrow B$:

$$\eta^x_C(g \circ f) = F(g \circ f)(x) = F(g)(F(f)(x)) = F(g)(\eta^x_B(f))$$

Quindi il diagramma commuta.

Verifichiamo che Φ e Ψ sono inverse:

- $\Phi(\Psi(x)) = \eta^x_A(\text{id}_A) = F(\text{id}_A)(x) = \text{id}_{\{F(A)\}}(x) = x$
- Per ogni η e ogni $f : A \rightarrow B$: $\Psi(\Phi(\eta))_B(f) = F(f)(\eta_A(\text{id}_A)) = \eta_B(f \circ \text{id}_A) = \eta_B(f)$ (per naturalità)

Quindi $\Phi \circ \Psi = \text{id}$ e $\Psi \circ \Phi = \text{id}$. ■

12.2 Categorie Cartesiane Chiuse

Le categorie cartesiane chiuse (CCC) forniscono la struttura matematica appropriata per modellare il λ -calcolo tipato e la semantica composizionale.

Definizione 12.2.1 (Prodotto categoriale): In una categoria C , il prodotto di oggetti A e B è un oggetto $A \times B$ insieme a morfismi (proiezioni):

$$\pi_1 : A \times B \rightarrow A \quad \pi_2 : A \times B \rightarrow B$$

tale che per ogni oggetto C e coppia di morfismi $f : C \rightarrow A$, $g : C \rightarrow B$, esiste un unico morfismo $\langle f, g \rangle : C \rightarrow A \times B$ tale che:

$$\pi_1 \circ \langle f, g \rangle = f \quad \pi_2 \circ \langle f, g \rangle = g$$

Questa è una proprietà universale: il prodotto è caratterizzato uniquely a meno di isomorfismo.

Definizione 12.2.2 (Oggetto terminale): Un oggetto 1 in C è terminale se per ogni oggetto A esiste esattamente un morfismo $! : A \rightarrow 1$.

Definizione 12.2.3 (Categoria cartesiana): Una categoria C è cartesiana se: (i) Ha tutti i prodotti finiti (ii) Ha un oggetto terminale 1

Definizione 12.2.4 (Esponenziale): In una categoria cartesiana C , l'esponenziale B^A (o $A \Rightarrow B$) di oggetti A e B è un oggetto insieme a un morfismo di valutazione:

$$\text{eval} : B^A \times A \rightarrow B$$

tale che per ogni oggetto C e morfismo $f : C \times A \rightarrow B$, esiste un unico morfismo $\lambda f : C \rightarrow B^A$ (il "curry" di f) tale che:

$$\text{eval} \circ (\lambda f \times \text{id}_A) = f$$

Questa proprietà è detta curry-uncurry adjunction o exponential transpose.

Definizione 12.2.5 (Categoria cartesiana chiusa): Una categoria C è cartesiana chiusa (CCC) se: (i) È cartesiana (ii) Per ogni coppia di oggetti A , B , esiste l'esponenziale B^A

Esempio 12.2.1 (CCC standard):

(i) **Set** è CCC:

- Prodotto = prodotto cartesiano di insiemi
- Terminale = insieme singoletto $\{*\}$
- Esponenziale B^A = insieme di tutte le funzioni da A a B

(ii) **ω -CPO** (ω -complete partial orders con mappe continue) è CCC:

- Prodotto = prodotto di poset (ordine componente per componente)
- Terminale = singoletto
- Esponenziale $[A \rightarrow B]$ = funzioni continue da A a B con ordine pointwise

(iii) Per ogni grammatica context-free G, la categoria **Parse(G)** ha:

- Oggetti = categorie sintattiche
- Morfismi = alberi di derivazione
- Prodotto dato dalla concatenazione

Teorema 12.2.1 (Leggi del λ -calcolo in CCC): In ogni CCC, valgono le seguenti identità (a meno di isomorfismo):

(i) $A \times 1 \cong A \cong 1 \times A$ (identità del prodotto) (ii) $A \times B \cong B \times A$ (simmetria) (iii) $(A \times B) \times C \cong A \times (B \times C)$ (associatività) (iv) $1^A \cong 1$ (v) $A^1 \cong A$ (vi) $(B \times C)^A \cong B^A \times C^A$ (vii) $B^{A \times C} \cong (B^A)^C$ (esponenziale transpose)

Dimostrazione (di vii, esponenziale transpose):

Dobbiamo costruire un isomorfismo $\varphi : B^{A \times C} \rightarrow (B^A)^C$.

Per ogni $f : 1 \rightarrow B^{A \times C}$, abbiamo $\text{eval} \circ (f \times \text{id}_{A \times C}) : 1 \times (A \times C) \rightarrow B$. Usando l'isomorfismo $1 \times (A \times C) \cong A \times C$, otteniamo $g : A \times C \rightarrow B$. Currificando rispetto a C: $\lambda_C(g) : A \rightarrow B^C$. Currificando rispetto a A: $\lambda_A(\lambda_C(g)) : 1 \rightarrow (B^A)^C$.

Definiamo $\varphi(f) = \lambda_A(\lambda_C(\text{eval} \circ (f \times \text{id}_{A \times C})))$.

Costruiamo l'inverso $\psi : (B^A)^C \rightarrow B^{A \times C}$ simmetricamente.

Verifichiamo $\varphi \circ \psi = \text{id}$ e $\psi \circ \varphi = \text{id}$ usando le proprietà universali del curry e uncurry. ■

Corollario 12.2.2: Ogni CCC fornisce un modello del λ -calcolo semplicemente tipato.

12.3 Lambda Calcolo Tipato

Il λ -calcolo tipato fornisce il fondamento formale per la semantica composizionale e corrisponde perfettamente alle CCC.

Definizione 12.3.1 (Tipi semplici): L'insieme dei tipi semplici Type è generato dalla grammatica:

$$\tau ::= b \mid \tau \rightarrow \tau \mid \tau \times \tau \mid 1$$

dove b varia su un insieme di tipi base.

Definizione 12.3.2 (Termini del λ -calcolo tipato): Dato un insieme numerabile di variabili tipate, i termini sono generati da:

$$t ::= x \mid \lambda x:\tau. t \mid t t \mid \langle t, t \rangle \mid \pi_1(t) \mid \pi_2(t) \mid *$$

dove * è l'unico termine di tipo 1.

Definizione 12.3.3 (Giudizi di tipo): I giudizi di tipo hanno la forma $\Gamma \vdash t : \tau$ dove:

- Γ è un contesto (lista finita di assunzioni $x_1:\tau_1, \dots, x_n:\tau_n$)
- t è un termine
- τ è un tipo

Regole di tipo:

(Var) $\frac{}{\Gamma \vdash x : \tau}$ se $(x:\tau) \in \Gamma$

(Abs) $\frac{\Gamma, x:\sigma \vdash t : \tau}{\Gamma \vdash \lambda x:\sigma. t : \sigma \rightarrow \tau}$

(App) $\frac{\Gamma \vdash t : \sigma \rightarrow \tau \quad \Gamma \vdash s : \sigma}{\Gamma \vdash t s : \tau}$

(Pair) $\frac{\Gamma \vdash t : \sigma \quad \Gamma \vdash s : \tau}{\Gamma \vdash \langle t, s \rangle : \sigma \times \tau}$

(Proj_i) $\frac{\Gamma \vdash t : \sigma \times \tau}{\Gamma \vdash \pi_i(t) : \sigma}$ (Proj₂) $\frac{\Gamma \vdash t : \sigma \times \tau}{\Gamma \vdash \pi_2(t) : \tau}$

(Unit) $\frac{}{\Gamma \vdash * : 1}$

Definizione 12.3.4 (Riduzione β): Le regole di riduzione β sono:

$(\beta \rightarrow) (\lambda x:\sigma. t) s \rightarrow_{\beta} t[s/x]$ (sostituzione di s per x in t) $(\beta \times_1) \pi_1(\langle t, s \rangle) \rightarrow_{\beta} t$ $(\beta \times_2) \pi_2(\langle t, s \rangle) \rightarrow_{\beta} s$

Teorema 12.3.1 (Soggetto riduzione - Type preservation): Se $\Gamma \vdash t : \tau$ e $t \rightarrow_{\beta} s$, allora $\Gamma \vdash s : \tau$.

Dimostrazione: Per induzione sulla derivazione di $t \rightarrow_{\beta} s$.

Caso $(\beta \rightarrow)$: Supponiamo $t = (\lambda x:\sigma. r)$ e $s = r[u/x]$. Per ipotesi $\Gamma \vdash (\lambda x:\sigma. r) u : \tau$. Quindi per inversione della regola (App):

- $\Gamma \vdash \lambda x:\sigma. r : \sigma \rightarrow \tau$
- $\Gamma \vdash u : \sigma$

Per inversione di (Abs): $\Gamma, x:\sigma \vdash r : \tau$.

Per il Lemma di Sostituzione (dimostrato separatamente): se $\Gamma, x:\sigma \vdash r : \tau$ e $\Gamma \vdash u : \sigma$, allora $\Gamma \vdash r[u/x] : \tau$.

Quindi $\Gamma \vdash s : \tau$. ■

Teorema 12.3.2 (Normalizzazione forte): Ogni termine ben tipato del λ -calcolo semplicemente tipato termina sotto qualsiasi sequenza di riduzioni β .

Dimostrazione (Schema - tramite reducibility candidates di Girard):

Definiamo per ogni tipo τ un insieme RED_τ di termini "riducibili" per induzione su τ :

- $RED_b = \{t : \vdash t : b \text{ e } t \text{ è fortemente normalizzante}\}$
- $RED_{\{\sigma \rightarrow \tau\}} = \{t : \vdash t : \sigma \rightarrow \tau \text{ e per ogni } s \in RED_\sigma, ts \in RED_\tau\}$
- $RED_{\{\sigma \times \tau\}} = \{t : \vdash t : \sigma \times \tau \text{ e } \pi_1(t) \in RED_\sigma \text{ e } \pi_2(t) \in RED_\tau\}$

Si dimostra che:

1. Ogni RED_τ contiene solo termini fortemente normalizzanti
2. Se $t \in RED_\tau$ e $t \rightarrow_\beta s$, allora $s \in RED_\tau$
3. Se t è neutrale (variabile o applicazione) e ogni suo ridotto è in RED_τ , allora $t \in RED_\tau$

Quindi ogni termine ben tipato è in qualche RED_τ , quindi fortemente normalizzante. ■

Teorema 12.3.3 (Corrispondenza Curry-Howard-Lambek): Esiste una corrispondenza tripartita tra:

- Dimostrazioni in logica intuizionista proposizionale
- Termini del λ -calcolo semplicemente tipato
- Morfismi in categorie cartesiane chiuse

Tabella di corrispondenza:

Logica	Lambda calcolo	Categoria
$A \wedge B$	$\sigma \times \tau$	$A \times B$
$A \rightarrow B$	$\sigma \rightarrow \tau$	B^A
$\top$	1	1
Dimostrazione di A	Termine di tipo A	Morfismo $1 \rightarrow A$
Modus ponens	Applicazione	Composizione con eval

Teorema 12.3.4: Ogni CCC dà luogo a un modello del λ -calcolo semplicemente tipato, e viceversa ogni λ -calcolo genera una CCC sintattica (la sua categoria term model).

12.4 Semantica Denotazionale

Definizione 12.4.1 (Interpretazione denotazionale): Data una CCC C , un'interpretazione denotazionale del λ -calcolo tipato in C assegna:

- Ad ogni tipo τ , un oggetto $\llbracket \tau \rrbracket \in \text{Ob}(C)$
- Ad ogni termine $\Gamma \vdash t : \tau$, un morfismo $\llbracket t \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket \tau \rrbracket$

dove $\llbracket x_1:\tau_1, \dots, x_n:\tau_n \rrbracket = \llbracket \tau_1 \rrbracket \times \dots \times \llbracket \tau_n \rrbracket$.

Regole semantiche:

$\llbracket b \rrbracket = B$ (oggetto base assegnato) $\llbracket \sigma \rightarrow \tau \rrbracket = \llbracket \tau \rrbracket^{\llbracket \sigma \rrbracket}$ $\llbracket \sigma \times \tau \rrbracket = \llbracket \sigma \rrbracket \times \llbracket \tau \rrbracket$ $\llbracket 1 \rrbracket = 1$ (oggetto terminale)

$\llbracket \Gamma \vdash x : \tau \rrbracket = \pi_i$ dove x è la i -esima variabile in Γ $\llbracket \Gamma \vdash \lambda x:\sigma. t : \sigma \rightarrow \tau \rrbracket = \text{curry}(\llbracket \Gamma, x:\sigma \vdash t : \tau \rrbracket)$ $\llbracket \Gamma \vdash t s : \tau \rrbracket = \text{eval} \circ \langle \llbracket \Gamma \vdash t : \sigma \rightarrow \tau \rrbracket, \llbracket \Gamma \vdash s : \sigma \rrbracket \rangle$ $\llbracket \Gamma \vdash \langle t, s \rangle : \sigma \times \tau \rrbracket = \langle \llbracket \Gamma \vdash t : \sigma \rrbracket, \llbracket \Gamma \vdash s : \tau \rrbracket \rangle$ $\llbracket \Gamma \vdash \pi_i(t) : \sigma \rrbracket = \pi_i \circ \llbracket \Gamma \vdash t : \sigma \times \tau \rrbracket$

Teorema 12.4.1 (Adeguatezza della semantica): Se $\Gamma \vdash t : \tau$ e $t \rightarrow_\beta s$ (dove $\rightarrow_\beta$ è la chiusura riflessiva-transitiva di $\rightarrow_\beta$), allora:

$$\llbracket \Gamma \vdash t : \tau \rrbracket = \llbracket \Gamma \vdash s : \tau \rrbracket$$

Dimostrazione: Basta mostrare che $\llbracket - \rrbracket$ preserva le riduzioni β singole. Per $\beta \rightarrow$:

$$\llbracket (\lambda x:\sigma. t) s \rrbracket = \text{eval} \circ \langle \text{curry}(\llbracket t \rrbracket), \llbracket s \rrbracket \rangle = \llbracket t \rrbracket \circ \langle \text{id}, \llbracket s \rrbracket \rangle \text{ (per la proprietà del curry)} = \llbracket t[s/x] \rrbracket \text{ (per il Lemma di Sostituzione semantico)}$$

Gli altri casi sono simili usando le leggi di β per prodotti. ■

Definizione 12.4.2 (Completezza della semantica): Un modello categoriale è completo se:

$$\forall \Gamma, \tau, t, s : \text{se } \llbracket \Gamma \vdash t : \tau \rrbracket = \llbracket \Gamma \vdash s : \tau \rrbracket, \text{ allora } \Gamma \vdash t = s \text{ (uguaglianza } \beta\text{-}\eta\text{)}$$

Teorema 12.4.2: La categoria Set è un modello completo del λ -calcolo semplicemente tipato.

Capitolo 13: Teoria dei Domini e Ricorsione

13.1 Partial Orders e Complete Partial Orders

Definizione 13.1.1 (Partially ordered set - poset): Un poset è una coppia $(D, \sqsubseteq)$ dove D è un insieme e $\sqsubseteq$ è una relazione binaria su D che è:

- (i) Riflessiva: $\forall x \in D: x \sqsubseteq x$ (ii) Transitiva: $\forall x, y, z \in D: (x \sqsubseteq y \wedge y \sqsubseteq z) \rightarrow x \sqsubseteq z$ (iii) Antisimmetrica: $\forall x, y \in D: (x \sqsubseteq y \wedge y \sqsubseteq x) \rightarrow x = y$

Definizione 13.1.2 (Catena - Chain): Una catena in un poset $(D, \sqsubseteq)$ è un sottoinsieme $C \subseteq D$ totalmente ordinato:

$$\forall x, y \in C: x \sqsubseteq y \vee y \sqsubseteq x$$

Definizione 13.1.3 (ω -catena): Una ω -catena è una catena numerabile crescente:

$$d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \dots$$

scritta come $(d_i)_{i \in \mathbb{N}}$.

Definizione 13.1.4 (Supremo): Dato un sottoinsieme $S \subseteq D$, un elemento $d \in D$ è il supremo (least upper bound) di S , scritto $d = \sqcup S$, se:

(i) $\forall x \in S: x \sqsubseteq d$ (d è upper bound) (ii) $\forall d' \in D: (\forall x \in S: x \sqsubseteq d') \rightarrow d \sqsubseteq d'$ (d è il least upper bound)

Definizione 13.1.5 (CPO - Complete Partial Order): Un CPO è un poset $(D, \sqsubseteq)$ dove:

(i) Esiste un elemento minimo $\perp \in D$ (bottom) tale che $\forall d \in D: \perp \sqsubseteq d$ (ii) Ogni ω -catena ha un supremo

Scriviamo $\sqcup_{i \in \mathbb{N}} d_i$ per il supremo della catena $(d_i)_{i \in \mathbb{N}}$.

Esempio 13.1.1:

(i) Qualsiasi insieme finito con $\perp$ e ordine discreto è un CPO (ii) $(\mathbb{N}_\perp, \leq)$ dove $\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$ con $\perp \leq n$ per ogni n (iii) L'insieme delle funzioni parziali $\mathbb{N} \rightarrow \mathbb{N}$ con ordine di estensione: $f \sqsubseteq g$ sse $\text{dom}(f) \subseteq \text{dom}(g)$ e $f(n) = g(n)$ per ogni $n \in \text{dom}(f)$

Definizione 13.1.6 (Funzione monotona): Una funzione $f: D \rightarrow E$ tra CPO è monotona se:

$$\forall x, y \in D: x \sqsubseteq y \rightarrow f(x) \sqsubseteq f(y)$$

Definizione 13.1.7 (Funzione continua): Una funzione $f: D \rightarrow E$ tra CPO è continua (o ω -continua) se è monotona e preserva supremi di ω -catene:

$$f(\sqcup_{i \in \mathbb{N}} d_i) = \sqcup_{i \in \mathbb{N}} f(d_i)$$

per ogni ω -catena $(d_i)_{i \in \mathbb{N}}$ in D .

Teorema 13.1.1: La composizione di funzioni continue è continua.

Dimostrazione: Siano $f: D \rightarrow E$ e $g: E \rightarrow F$ continue, e sia (d_i) una ω -catena in D . Allora:

$$(g \circ f)(\sqcup d_i) = g(f(\sqcup d_i)) = g(\sqcup f(d_i)) \text{ (f continua)} = \sqcup g(f(d_i)) \text{ (g continua)} = \sqcup (g \circ f)(d_i)$$

Quindi $g \circ f$ è continua. ■

Definizione 13.1.8 (Categoria CPO): La categoria **CPO** ha:

- Oggetti = CPO
- Morfismi = funzioni continue

Per le dimostrazioni precedenti, **CPO** è effettivamente una categoria.

13.2 Teorema del Punto Fisso

Teorema 13.2.1 (Teorema del Punto Fisso di Kleene-Tarski): Sia D un CPO e $f : D \rightarrow D$ una funzione continua. Allora f ha un minimo punto fisso $\text{fix}(f)$ dato da:

$$\text{fix}(f) = \sqcup_{n \in \mathbb{N}} f^n(\perp)$$

dove f^n denota la n -esima iterazione di f : $f^0(\perp) = \perp$, $f^{n+1}(\perp) = f(f^n(\perp))$.

Dimostrazione:

Passo 1: La sequenza $(f^n(\perp))$ è una ω -catena.

Per induzione su n :

- Base: $f^0(\perp) = \perp \sqsubseteq f(\perp) = f^1(\perp)$ poiché $\perp$ è minimo
- Passo: Se $f^n(\perp) \sqsubseteq f^{n+1}(\perp)$, allora $f^{n+1}(\perp) = f(f^n(\perp)) \sqsubseteq f(f^{n+1}(\perp)) = f^{n+2}(\perp)$ per monotonia di f

Passo 2: Definiamo $d = \sqcup_{n \in \mathbb{N}} f^n(\perp)$. Mostriamo che $f(d) = d$.

$$f(d) = f(\sqcup_{n \in \mathbb{N}} f^n(\perp)) = \sqcup_{n \in \mathbb{N}} f(f^n(\perp)) \quad (f \text{ continua}) = \sqcup_{n \in \mathbb{N}} f^{n+1}(\perp) = \sqcup_{n \in \mathbb{N}} f^n(\perp) \quad (\text{l'elemento } f^0(\perp) = \perp \text{ è sotto tutti gli altri}) = d$$

Passo 3: Mostriamo che d è il minimo punto fisso.

Sia d' un punto fisso di f , ovvero $f(d') = d'$. Dimostriamo per induzione che $f^n(\perp) \sqsubseteq d'$ per ogni n :

- Base: $f^0(\perp) = \perp \sqsubseteq d'$ poiché $\perp$ è minimo
- Passo: Se $f^n(\perp) \sqsubseteq d'$, allora $f^{n+1}(\perp) = f(f^n(\perp)) \sqsubseteq f(d') = d'$ per monotonia

Quindi $d = \sqcup_{n \in \mathbb{N}} f^n(\perp) \sqsubseteq d'$ poiché ogni $f^n(\perp) \sqsubseteq d'$. ■

Corollario 13.2.2: Ogni funzione continua $f : D \rightarrow D$ ha un unico minimo punto fisso.

Esempio 13.2.1: Consideriamo la funzione ricorsiva:

$$\text{fact}(n) = \text{if } n = 0 \text{ then } 1 \text{ else } n \times \text{fact}(n-1)$$

Modelliamo fact come un punto fisso. Definiamo $F : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$ da:

$$F(g)(n) = \text{if } n = 0 \text{ then } 1 \text{ else } n \times g(n-1)$$

Il minimo punto fisso di F è la funzione fattoriale:

$$\text{fact} = \text{fix}(F) = \sqcup_{i \in \mathbb{N}} F^i(\perp)$$

dove:

- $F^0(\perp) = \perp$ (funzione totalmente indefinita)

- $F^1(\perp)(n)$ = definita solo per $n=0$, valore 1
- $F^2(\perp)(n)$ = definita per $n \in \{0,1\}$, valori $\{1,1\}$
- $F^3(\perp)(n)$ = definita per $n \in \{0,1,2\}$, valori $\{1,1,2\}$
- ...

Teorema 13.2.3 (Induzione sul punto fisso - Park induction): Sia $f : D \rightarrow D$ continua con $\text{fix}(f) = d$, e sia $P \subseteq D$ un insieme ammissibile (chiuso sotto supremi di catene). Se:

(i) $\perp \in P$ (ii) $\forall x \in P: f(x) \in P$

allora $d \in P$.

Dimostrazione: Per ipotesi (i), $f^0(\perp) = \perp \in P$. Per induzione e ipotesi (ii), $f^n(\perp) \in P$ per ogni n . Poiché P è ammissibile, $\sqcup_{n \in \mathbb{N}} f^n(\perp) = d \in P$. ■

13.3 Costruzione di Domini per Tipi Ricorsivi

Definizione 13.3.1 (Tipo ricorsivo): Un tipo ricorsivo ha la forma:

$\mu X. T(X)$

dove T è un costruttore di tipo (funttore). L'interpretazione dovrebbe soddisfare:

$$\llbracket \mu X. T(X) \rrbracket \cong T(\llbracket \mu X. T(X) \rrbracket)$$

ovvero, il tipo è isomorfo alla sua "espansione".

Esempio 13.3.1: Liste di interi:

$$\text{List} = \mu X. 1 + (\mathbb{Z} \times X)$$

che dice: una lista è o vuota (1) o una coppia di un intero e una lista.

Teorema 13.3.1: Se $T : \text{CPO} \rightarrow \text{CPO}$ è un funttore localmente continuo, allora esiste un CPO D tale che $D \cong T(D)$.

Dimostrazione (Schema - approccio via limiti inversi):

Costruiamo una sequenza decrescente di CPO e proiezioni:

$$D_0 \xleftarrow{p_1} D_1 \xleftarrow{p_2} D_2 \xleftarrow{p_3} \dots$$

dove $D_{i+1} = T(D_i)$ e $p_{i+1} : T(D_i) \rightarrow D_i$ è definita usando $T(p_i)$.

Il limite inverso $\lim_{\leftarrow i} D_i$ esiste in CPO e soddisfa:

$$\lim_{\leftarrow i} D_i \cong T(\lim_{\leftarrow i} D_i)$$

Questo costruisce il punto fisso del funttore T a livello di oggetti. ■

Esempio 13.3.2: Per il tipo Stream di stream infiniti di $\mathbb{N}$:

Stream = $\mu X. \mathbb{N} \times X$

Il CPO Stream può essere costruito come limite inverso della sequenza:

$$1 \leftarrow \mathbb{N} \leftarrow \mathbb{N} \times \mathbb{N} \leftarrow \mathbb{N} \times \mathbb{N} \times \mathbb{N} \leftarrow \dots$$

dove ogni proiezione dimentica l'ultima componente.

Trattato sulla Teoria Formale delle Lingue Naturali

Parte IX: Automi Avanzati e Teoria dell'Apprendimento

Capitolo 14: Automi Pesati e Probabilistici

14.1 Automi Pesati

Gli automi pesati generalizzano gli automi a stati finiti assegnando pesi (da un semiring) alle transizioni, permettendo di modellare fenomeni quantitativi nel linguaggio.

Definizione 14.1.1 (Semiring): Un semiring è una quintupla $(K, \oplus, \otimes, 0, 1)$ dove:

- (i) $(K, \oplus, 0)$ è un monoide commutativo (addizione)
- (ii) $(K, \otimes, 1)$ è un monoide (moltiplicazione)
- (iii) $\otimes$ distribuisce su $\oplus$: $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- (iv) 0 è annichilatore per $\otimes$: $0 \otimes a = a \otimes 0 = 0$

Esempio 14.1.1 (Semiring standard):

- (i) **Booleano**: $(\{0,1\}, \vee, \wedge, 0, 1)$ - per automi non deterministici standard
- (ii) **Tropicale**: $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$ - per cammini più corti
- (iii) **Probabilistico**: $([0,1], +, \times, 0, 1)$ - con + limitato a somme ≤ 1
- (iv) **Conteggio**: $(\mathbb{N} \cup \{\infty\}, +, \times, 0, 1)$ - per contare cammini
- (v) **Log**: $(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$ - per probabilità log-space

Definizione 14.1.2 (Automa a stati finiti pesato): Un automa pesato su semiring K è una quintupla $A = (Q, \Sigma, I, F, \delta)$ dove:

- (i) Q è un insieme finito di stati
- (ii) Σ è un alfabeto finito
- (iii) $I : Q \rightarrow K$ assegna pesi iniziali agli stati
- (iv) $F : Q \rightarrow K$ assegna pesi finali agli stati
- (v) $\delta : Q \times \Sigma \times Q \rightarrow K$ è la funzione di transizione pesata

Definizione 14.1.3 (Peso di un cammino): Un cammino π in A è una sequenza:

$$\pi = (q_0, a_1, q_1)(q_1, a_2, q_2) \dots (q_{n-1}, a_n, q_n)$$

Il peso del cammino è:

$$\text{weight}(\pi) = l(q_0) \otimes \delta(q_0, a_1, q_1) \otimes \delta(q_1, a_2, q_2) \otimes \dots \otimes \delta(q_{n-1}, a_n, q_n) \otimes F(q_n)$$

Definizione 14.1.4 (Serie formale riconosciuta): La serie formale riconosciuta da A è la funzione $S : \Sigma^* \rightarrow K$ definita da:

$$S(w) = \oplus_{\pi: \pi \text{ è un cammino che legge } w} \text{weight}(\pi)$$

dove la somma è su tutti i cammini che leggono la stringa w .

Teorema 14.1.1 (Rappresentazione matriciale): Un automa pesato A può essere rappresentato mediante matrici:

- Vettore iniziale $I \in K^{|Q|}$
- Matrice di transizione $M_a \in K^{|Q| \times |Q|}$ per ogni $a \in \Sigma$
- Vettore finale $F \in K^{|Q|}$

dove $[M_a]_{i,j} = \delta(q_i, a, q_j)$.

Allora per $w = a_1 a_2 \dots a_n$:

$$S(w) = I^T M_{a_1} M_{a_2} \dots M_{a_n} F$$

Dimostrazione: La moltiplicazione matriciale calcola esattamente la somma sui cammini. L'elemento (i,j) di $M_{a_1} M_{a_2} \dots M_{a_n}$ è:

$$\sum_{q_1, \dots, q_{n-1}} M_{a_1}[i, q_1] \otimes M_{a_2}[q_1, q_2] \otimes \dots \otimes M_{a_n}[q_{n-1}, j]$$

che somma i pesi di tutti i cammini da q_i a q_j leggendo w . Moltiplicando per I^T e F si ottengono i pesi iniziali e finali. ■

Definizione 14.1.5 (Chiusura di Kleene per semiring): Per una matrice $M \in K^{n \times n}$, la chiusura di Kleene è:

$$M^* = \oplus_{i=0}^{\infty} M^i = I \oplus M \oplus M^2 \oplus M^3 \oplus \dots$$

quando questa serie converge nel semiring.

Teorema 14.1.2 (Algoritmo di chiusura per semiring k-chiusi): In un semiring k-chiuso (dove sums finite di prodotti esistono), M^* può essere calcolato mediante l'algoritmo di Floyd-Warshall generalizzato in tempo $O(n^3)$.

Dimostrazione (Schema): L'algoritmo calcola iterativamente $M^{(k)}$ dove $M^{(k)}_{ij}$ è la somma dei pesi di tutti i cammini da i a j passando solo per stati con indice $\leq k$:

$$M^{(k)}_{ij} = M^{(k-1)}_{ij} \oplus (M^{(k-1)}_{ik} \otimes (M^{(k-1)}_{kk})^* \otimes M^{(k-1)}_{kj})$$

dove $(M^{(k-1)}_{kk})^*$ è la chiusura dell'elemento singolo. ■

14.2 Automi Probabilistici

Definizione 14.2.1 (Automa probabilistico a stati finiti - PFA): Un PFA è un automa pesato sul semiring probabilitistico dove:

(i) $\sum_{q \in Q} I(q) = 1$ (distribuzione iniziale) (ii) $\sum_{q' \in Q} \delta(q, a, q') = 1$ per ogni q, a (transizioni stocastiche) (iii) $F(q) \in \{0, 1\}$ (stati accettanti deterministici)

Interpretazione: $I(q)$ è la probabilità di iniziare nello stato q , e $\delta(q, a, q')$ è la probabilità di transire da q a q' leggendo a .

Definizione 14.2.2 (Probabilità di accettazione): La probabilità che un PFA A accetti una stringa w è:

$$P_A(w) = \sum_{\pi \text{ accettante per } w} \Pr(\pi)$$

dove $\Pr(\pi)$ è il prodotto dei pesi lungo il cammino (interpretati come probabilità).

Definizione 14.2.3 (Linguaggio stocastico): Un linguaggio stocastico è una distribuzione di probabilità $L : \Sigma^* \rightarrow [0, 1]$ tale che $\sum_{w \in \Sigma^*} L(w) = 1$.

Teorema 14.2.1: I PFA definiscono esattamente la classe delle distribuzioni probabilitistiche razionali su Σ^* .

Definizione 14.2.4 (Catena di Markov nascosta - HMM): Un HMM è una sestupla $H = (Q, \Sigma, A, B, \pi, F)$ dove:

(i) Q è un insieme finito di stati nascosti (ii) Σ è un alfabeto di simboli osservabili (iii) $A : Q \times Q \rightarrow [0, 1]$ è la matrice di transizione degli stati: $A_{ij} = \Pr(q_{t+1} = j \mid q_t = i)$ (iv) $B : Q \times \Sigma \rightarrow [0, 1]$ è la matrice di emissione: $B_{i,a} = \Pr(O_t = a \mid q_t = i)$ (v) $\pi : Q \rightarrow [0, 1]$ è la distribuzione iniziale (vi) $F : Q \rightarrow \{0, 1\}$ indica stati finali

Relazione con PFA: Un HMM è equivalente a un PFA dove le transizioni hanno la forma:

$$\delta(q, a, q') = A_{q,q'} \times B_{q',a}$$

Problema 14.2.1 (Problemi fondamentali per HMM):

(i) **Evaluation:** Data una sequenza di osservazioni $O = o_1 \dots o_T$, calcolare $\Pr(O \mid H)$ (ii) **Decoding:** Data O , trovare la sequenza di stati nascosti più probabile (iii) **Learning:** Dati esempi di osservazioni, stimare i parametri A, B, π

Algoritmo 14.2.1 (Forward algorithm per Evaluation):

Definiamo $\alpha_t(i) = \Pr(o_1 \dots o_t, q_t = i \mid H)$.

Ricorsione:

- Inizializzazione: $\alpha_1(i) = \pi_i \times B_{i,o_1}$
- Ricorsione: $\alpha_{t+1}(j) = B_{j,o_{t+1}} \times \sum_i \alpha_t(i) \times A_{ij}$
- Terminazione: $\Pr(O \mid H) = \sum_i \alpha_T(i) \times F(i)$

Complessità: $O(T \times |Q|^2)$

Algoritmo 14.2.2 (Viterbi algorithm per Decoding):

Definiamo $v_t(i) = \max_{q_1 \dots q_{t-1}} \Pr(q_1 \dots q_{t-1}, q_t = i, o_1 \dots o_t \mid H)$.

Ricorsione:

- Inizializzazione: $v_1(i) = \pi_i \times B_{\{i, o_1\}}$
- Ricorsione: $v_{t+1}(j) = B_{\{j, o_{t+1}\}} \times \max_i (v_t(i) \times A_{\{ij\}})$
- Backtracking: ricostruire il cammino ottimale

Complessità: $O(T \times |Q|^2)$

Algoritmo 14.2.3 (Baum-Welch algorithm per Learning):

Algoritmo EM (Expectation-Maximization):

E-step: Calcola statistiche attese sotto il modello corrente

- $\xi_t(i, j) = \Pr(q_t = i, q_{t+1} = j \mid O, H)$
- $\gamma_t(i) = \Pr(q_t = i \mid O, H)$

M-step: Aggiorna i parametri per massimizzare la likelihood

- $A'_{\{ij\}} = (\sum_t \xi_t(i, j)) / (\sum_t \gamma_t(i))$
- $B'_{\{i, a\}} = (\sum_{t: o_t=a} \gamma_t(i)) / (\sum_t \gamma_t(i))$
- $\pi'_i = \gamma_1(i)$

Itera fino a convergenza.

Teorema 14.2.2: L'algoritmo Baum-Welch converge a un massimo locale della likelihood.

Dimostrazione (Schema): Dimostrare che ogni iterazione EM aumenta (o mantiene costante) la log-likelihood. Questo segue dalla proprietà generale degli algoritmi EM che la funzione Q aumenta ad ogni passo. ■

14.3 Automi Quantistici

Gli automi quantistici sfruttano i principi della meccanica quantistica per il riconoscimento dei linguaggi.

Definizione 14.3.1 (Automa quantistico a stati finiti - QFA): Un QFA è una quintupla $Q = (Q, \Sigma, \{U_\sigma\}_{\sigma \in \Sigma \cup \{\epsilon\}}, q_0, Q_{\text{acc}})$ dove:

- (i) Q è un insieme finito di stati (base ortonormale di uno spazio di Hilbert $\mathbb{C}^{|Q|}$)
- (ii) Σ è l'alfabeto di input
- (iii) $U_\sigma : \mathbb{C}^{|Q|} \rightarrow \mathbb{C}^{|Q|}$ è un operatore unitario per ogni $\sigma \in \Sigma \cup \{\epsilon\}$
- (iv) $q_0 \in Q$ è lo stato iniziale
- (v) $Q_{\text{acc}} \subseteq Q$ sono gli stati accettanti

Definizione 14.3.2 (Operatore unitario): Un operatore $U : \mathbb{C}^n \rightarrow \mathbb{C}^n$ è unitario se $U^\dagger U = U U^\dagger = I$, dove $U^\dagger$ è l'aggiunto coniugato di U .

Equivalentemente, U preserva il prodotto interno: $\langle Uv, Uw \rangle = \langle v, w \rangle$.

Definizione 14.3.3 (Evoluzione quantistica): Dato un input $w = a_1 a_2 \dots a_n$, lo stato quantistico evolve come:

$$|\psi\rangle = U_{a_n} U_{a_{n-1}} \dots U_{a_2} U_{a_1} |q_0\rangle$$

dove $|q_0\rangle$ è il vettore di base corrispondente allo stato iniziale.

Definizione 14.3.4 (Probabilità di accettazione): La probabilità di accettare w è:

$$P_Q(w) = \sum_{q \in Q_{acc}} |\langle q | \psi \rangle|^2$$

dove $|\langle q | \psi \rangle|^2$ è il quadrato del modulo dell'ampiezza dello stato q .

Esempio 14.3.1 (QFA per $\{a^n b^n : n \geq 0\}$):

Contrariamente agli automi classici, è stato dimostrato che i QFA measure-once non possono riconoscere tutti i linguaggi regolari. Tuttavia, i QFA measure-many (che permettono misurazioni intermedie) sono più potenti.

Teorema 14.3.1 (Limitazioni dei QFA measure-once): I linguaggi riconosciuti da QFA measure-once con probabilità limitata (bounded error) formano una sottoclasse propria dei linguaggi regolari.

Dimostrazione (Schema - via proprietà di mixing): Un QFA measure-once su input lungo causa interferenza quantistica che porta a una distribuzione di probabilità essenzialmente uniforme sugli stati, rendendo impossibile distinguere certe stringhe. ■

Definizione 14.3.5 (QFA con errore limitato): Un QFA riconosce un linguaggio L con errore ϵ se:

- $\forall w \in L: P_Q(w) \geq 1 - \epsilon$
- $\forall w \notin L: P_Q(w) \leq \epsilon$

Teorema 14.3.2: Esistono linguaggi regolari che non possono essere riconosciuti da QFA measure-once con errore limitato $< 1/2$.

Teorema 14.3.3: I QFA measure-many (con misurazioni intermedie) possono simulare DFA, quindi riconoscono tutti i linguaggi regolari.

Definizione 14.3.6 (Vantaggio quantistico): In alcuni casi, i QFA possono essere esponenzialmente più compatti dei DFA corrispondenti. Per esempio, il linguaggio $\text{Mod}_p = \{w : |w| \equiv 0 \pmod{p}\}$ richiede p stati per un DFA ma può essere riconosciuto da un QFA con $O(\log p)$ stati.

Capitolo 15: Teoria dell'Apprendimento Grammaticale

15.1 Paradigma PAC Learning

Definizione 15.1.1 (PAC Learning - Probably Approximately Correct): Un concetto $c : X \rightarrow \{0,1\}$ su uno spazio di esempi X è PAC-learnable da una classe di ipotesi H se esiste un algoritmo A tale che:

Per ogni distribuzione D su X , per ogni $\epsilon, \delta > 0$, dato un numero polinomiale di esempi etichettati secondo c e D , A produce con probabilità $\geq 1-\delta$ un'ipotesi $h \in H$ tale che:

$$\Pr_{x \sim D}[h(x) \neq c(x)] \leq \epsilon$$

dove la probabilità è rispetto a D e alla randomizzazione di A .

Definizione 15.1.2 (Sample complexity): La sample complexity $m(\epsilon, \delta)$ è il numero minimo di esempi necessari affinché l'algoritmo garantisca ϵ -accuratezza con probabilità $\geq 1-\delta$.

Definizione 15.1.3 (Dimensione VC - Vapnik-Chervonenkis): La dimensione VC di una classe di ipotesi H , denotata $VC(H)$, è la massima cardinalità di un insieme $S \subseteq X$ tale che H shatter S , ovvero:

$$|\{h \cap S : h \in H\}| = 2^{|S|}$$

Se nessun insieme finito massimale esiste, $VC(H) = \infty$.

Teorema 15.1.1 (Teorema fondamentale del PAC learning): Una classe H è PAC-learnable se e solo se $VC(H) < \infty$. Inoltre, la sample complexity è:

$$m(\epsilon, \delta) = O((VC(H)/\epsilon) \times \log(1/\delta))$$

Dimostrazione (Schema):

($\Leftarrow$) Se $VC(H) = d < \infty$, allora per il Lemma di Sauer:

$$|\{h \cap S : h \in H\}| \leq \sum_{i=0}^d \binom{|S|}{i} \leq (e|S|/d)^d$$

Usando uniform convergence bounds, si dimostra che con $m \geq O((d/\epsilon) \log(1/\delta))$ esempi, con alta probabilità ogni h consistente con i dati ha errore $\leq \epsilon$.

($\Rightarrow$) Se $VC(H) = \infty$, allora H può shatter insiemi arbitrariamente grandi, e nessun numero finito di esempi può garantire convergenza uniforme. ■

Esempio 15.1.1 (Dimensione VC di classi di linguaggi):

(i) **Linguaggi finiti**: $VC = 0$ (ogni concetto è una singola stringa) (ii) **k-DNF** (formule booleane in forma normale disgiuntiva con clausole di dimensione $\leq k$): $VC = O(n^k)$ dove n è il numero di variabili (iii) **DFA con k stati**: $VC = O(k \log k)$

Teorema 15.1.2 (Lower bound per DFA): Imparare DFA nel modello PAC è difficile (computazionalmente). Più precisamente, imparare anche solo l'esistenza di un DFA di dimensione k è NP-hard.

15.2 Grammatical Inference

Definizione 15.2.1 (Problema di identificazione nel limite - Gold): Una classe di linguaggi L è identificabile nel limite da un algoritmo A se:

Per ogni $L \in \mathcal{L}$ e ogni presentazione (enumerazione) di esempi positivi di L , esiste n_0 tale che per ogni $n \geq n_0$:

- A produce un'ipotesi G_n
- $L(G_n) = L$

Teorema 15.2.1 (Teorema di Gold): Nessuna classe superfinita di linguaggi enumerabili è identificabile nel limite da solo esempi positivi.

Dimostrazione: Supponiamo che L sia identificabile nel limite. Consideriamo due linguaggi $L_1, L_2 \in \mathcal{L}$ con $L_1 \subset L_2$. Sia $s_1, s_2, \dots$ un'enumerazione degli elementi di L_1 . Poiché A identifica L_1 , esiste n_0 tale che dopo aver visto $s_1, \dots, s_{n_0}$, A converge a una grammatica G con $L(G) = L_1$.

Ma la stessa sequenza $s_1, \dots, s_{n_0}$ è anche un prefisso di un'enumerazione di L_2 . Se presentiamo poi elementi di $L_2 \setminus L_1$, A deve cambiare ipotesi. Ma non può mai essere sicuro se ha visto tutti gli elementi di L_1 o se appartiene a L_2 , quindi non può convergere definitivamente. Contraddizione. ■

Corollario 15.2.2: La classe di tutti i linguaggi regolari non è identificabile nel limite da soli esempi positivi.

Definizione 15.2.2 (Identificazione da esempi positivi e negativi): L è identificabile da esempi positivi e negativi se esiste un algoritmo che, data una presentazione di esempi $x_1, x_2, \dots$ con etichette indicanti se $x_i \in L$ o $x_i \notin L$, converge a una grammatica corretta.

Teorema 15.2.3: La classe dei linguaggi regolari è identificabile nel limite da esempi positivi e negativi.

Dimostrazione (Schema - algoritmo RPNI): L'algoritmo Regular Positive and Negative Inference costruisce un automa prefix tree dai dati positivi, poi lo generalizza unendo stati compatibili con i dati negativi. Si può dimostrare convergenza nel limite. ■

Definizione 15.2.3 (k-reversibile): Un DFA è k-reversibile se identificare gli stati basandosi sui k simboli precedenti (invece dei k simboli successivi) produce lo stesso automa.

Teorema 15.2.4: La classe dei linguaggi k-reversibili è identificabile in tempo polinomiale da esempi positivi e negativi.

Algoritmo 15.2.1 (Algoritmo di Angluin per k-reversibili):

Input: Esempi positivi S^+ , esempi negativi S^- Output: DFA k-reversibile consistente

1. Costruire il prefix tree acceptor per S^+
2. Per ogni coppia di stati (p, q) :
 - Se merging p e q è consistente con S^- e preserva k-reversibilità, merge
3. Restituire il DFA risultante

Teorema 15.2.5 (Teorema di Angluin L^*): Esiste un algoritmo di apprendimento attivo (con query) per DFA che apprende un DFA minimo con n stati in tempo polinomiale in n e nella lunghezza del controesempio più lungo.

L'algoritmo usa due tipi di query:

- **Membership query:** " $w \in L?$ "
- **Equivalence query:** " $L(H) = L?$ " dove H è un'ipotesi; se no, restituisce un controesempio

Dimostrazione (Schema dell'algoritmo L^*):

1. Mantenere una observation table (S, E, T) dove:
 - S è un insieme di prefissi
 - E è un insieme di suffissi
 - $T(s \cdot e)$ indica se $s \cdot e \in L$ (ottenuto via membership queries)
2. Assicurare che la tabella sia chiusa e consistente
3. Costruire un'ipotesi H dalla tabella
4. Fare equivalence query
5. Se H è corretta, fermarsi; altrimenti usare il controesempio per raffinare S ed E

Complessità: $O(|\Sigma| \times n^2 \times m)$ membership queries, $O(n)$ equivalence queries, dove m è la lunghezza del controesempio più lungo. ■

15.3 MDL e Compressione

Definizione 15.3.1 (Minimum Description Length - MDL): Il principio MDL afferma che la migliore ipotesi per spiegare i dati è quella che minimizza:

$$L(H) + L(D | H)$$

dove:

- $L(H)$ è la lunghezza della descrizione dell'ipotesi H
- $L(D | H)$ è la lunghezza della descrizione dei dati D dato H

Teorema 15.3.1 (Equivalenza con Bayesiano): MDL è equivalente a massimizzare la probabilità a posteriori con una prior basata sulla lunghezza della descrizione:

$$P(H | D) \propto P(D | H) \times P(H) = 2^{-L(D|H)} \times 2^{-L(H)}$$

Definizione 15.3.2 (Complessità di Kolmogorov): La complessità di Kolmogorov di una stringa x è:

$$K(x) = \min\{|p| : U(p) = x\}$$

dove U è una macchina di Turing universale e p è un programma.

Teorema 15.3.2 (Incomputabilità di K): La funzione K non è computabile.

Dimostrazione: Supponiamo K computabile. Costruiamo il seguente programma paradossale:

"Trova la prima stringa x con $K(x) > |\text{questo programma}|$ "

Sia l la lunghezza di questo programma. Il programma produce una stringa x con $K(x) > l$. Ma allora abbiamo descritto x con un programma di lunghezza l , quindi $K(x) \leq l$. Contraddizione. ■

Teorema 15.3.3 (Teorema di codifica di Shannon): Per una sorgente con entropia H , il numero atteso di bit per simbolo in qualsiasi codifica senza perdita è $\geq H$.

Applicazione 15.3.1: Per linguaggi regolari, il MDL favorisce DFA più piccoli, fornendo una giustificazione information-theoretic per la minimizzazione di automi.

15.4 Statistical Language Models

Definizione 15.4.1 (n -gram model): Un modello n -gram stima la probabilità di una stringa come:

$$P(w_1 w_2 \dots w_m) = \prod_{i=1}^m P(w_i | w_{i-n+1} \dots w_{i-1})$$

Stima MLE (Maximum Likelihood Estimation):

$$\hat{P}(w_i | w_{i-n+1} \dots w_{i-1}) = \text{Count}(w_{i-n+1} \dots w_i) / \text{Count}(w_{i-n+1} \dots w_{i-1})$$

Problema: Molti n -gram hanno count zero (data sparsity).

Soluzione 15.4.1 (Smoothing techniques):

(i) **Add-k smoothing (Laplace):** $P(w_i | \text{context}) = (\text{Count}(\text{context}, w_i) + k) / (\text{Count}(\text{context}) + k|V|)$

(ii) **Good-Turing smoothing:** Ridistribuire la massa di probabilità dai n -gram frequenti a quelli rari basandosi su conteggi di conteggi

(iii) **Kneser-Ney smoothing:** Interpolazione sofisticata che usa continuation probability

Definizione 15.4.2 (Perplexity): La perplexity di un modello linguistico su un test set è:

$$PP(W) = P(w_1 \dots w_N)^{-1/N} = N \sqrt[N]{1 / P(w_1 \dots w_N)}$$

Intuitivamente, perplexity è il numero medio di scelte equiprobabili per la prossima parola.

Teorema 15.4.1: Minimizzare perplexity è equivalente a massimizzare log-likelihood sul test set.

Dimostrazione: $\log PP(W) = -(1/N) \log P(W)$

Quindi minimizzare PP equivale a minimizzare $-\log P(W)$, ovvero massimizzare $\log P(W)$. ■

Definizione 15.4.3 (Cross-entropy): Il cross-entropy rate tra una distribuzione vera p e un modello q è:

$$H(p, q) = \lim_{n \rightarrow \infty} -(1/n) \sum_{\{w: |w|=n\}} p(w) \log q(w)$$

Teorema 15.4.2 (Shannon-McMillan-Breiman) - Continuazione:

Per un processo stocastico ergodico: $H(p, q) = \lim_{n \rightarrow \infty} -(1/n) \log q(W_n)$

quasi certamente, dove W_n è una sequenza generata da p .

Dimostrazione: Il teorema è un'estensione del teorema AEP (Asymptotic Equipartition Property). Per un processo ergodico con distribuzione p e modello q :

1. Per la legge dei grandi numeri ergodica: $-(1/n) \log p(W_n) \rightarrow H(p)$ quasi certamente
2. Similarmente per il cross-entropy: $-(1/n) \log q(W_n) \rightarrow H(p, q)$ quasi certamente
3. La convergenza segue dall'ergodicità che garantisce che le medie temporali convergano alle medie di insieme. ■

Teorema 15.4.3 (Disuguaglianza di Kraft-McMillan): Per un codice univocamente decifrabile con lunghezze di codice $l_1, l_2, \dots, l_n$ su alfabeto di dimensione D : $\sum_{i=1}^n D^{-l_i} \leq 1$

Applicazione alla modellazione linguistica: Un modello linguistico può essere visto come un codice, e la disuguaglianza di Kraft fornisce limiti sulla compressione ottenibile.

15.5 Neural Language Models

Definizione 15.5.1 (Recurrent Neural Network Language Model): Un RNN-LM è definito da:

- $h_t = \sigma(W_h h_{t-1} + W_x x_t + b_h)$
- $y_t = \text{softmax}(W_y h_t + b_y)$
- $P(w_t | w_1 \dots w_{t-1}) = y_t[w_t]$

dove h_t è lo stato nascosto, x_t è l'embedding della parola w_t , e y_t è la distribuzione di probabilità sul vocabolario.

Teorema 15.5.1 (Capacità espressiva delle RNN): Le RNN con funzioni di attivazione saturanti (sigmoid, tanh) sono Turing-complete in linea di principio, ma con precisione finita sono limitate a riconoscere linguaggi regolari.

Dimostrazione (Schema): Con precisione reale, una RNN può simulare un automa a pila (PDA):

- Gli stati nascosti codificano la pila
- Le transizioni non lineari permettono operazioni push/pop

Tuttavia, con precisione finita (floating point), lo spazio degli stati è finito, quindi solo linguaggi regolari sono riconoscibili in pratica. ■

Definizione 15.5.2 (Long Short-Term Memory - LSTM): Un LSTM estende le RNN con gates:

- Input gate: $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$
- Forget gate: $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$
- Output gate: $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$
- Cell state: $C_t = f_t \odot C_{t-1} + i_t \odot \tanh(W_C[h_{t-1}, x_t] + b_C)$
- Hidden state: $h_t = o_t \odot \tanh(C_t)$

Teorema 15.5.2 (Gradiente nelle LSTM): Le LSTM mitigano il problema del vanishing/exploding gradient attraverso il meccanismo della cell state che permette flusso diretto del gradiente.

Definizione 15.5.3 (Transformer Language Model): Un Transformer utilizza self-attention: $\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k})V$

dove Q, K, V sono query, key, value matrices.

Teorema 15.5.3 (Approssimazione universale dei Transformer): I Transformer con sufficiente profondità e larghezza possono approssimare qualsiasi funzione sequenza-a-sequenza continua su dominio compatto.

15.6 Teoremi di Convergenza per l'Apprendimento

Teorema 15.6.1 (No Free Lunch per la Classificazione): Non esiste un algoritmo di apprendimento universalmente ottimale. Per ogni algoritmo A, esiste una distribuzione D per cui A fallisce.

Dimostrazione formale: Sia U l'insieme di tutti i possibili target concepts. Per ogni algoritmo A e prior P su U: $E_{c \sim P}[\text{error}(A, c)] \geq \min_A E_{c \sim P}[\text{error}(A, c)]$

Ma per ogni A, possiamo costruire un prior avversariale P_A tale che A performa male. ■

Teorema 15.6.2 (Convergenza PAC per CFG): La classe delle CFG non è PAC-learnable da soli esempi positivi in tempo polinomiale (assumendo $RP \neq NP$).

Dimostrazione: Riduzione da Graph Isomorphism o problemi di fattorizzazione. L'apprendimento di CFG richiede la soluzione di problemi computazionalmente hard. ■

Teorema 15.6.3 (Bound di Hoeffding per stimatori di frequenza): Sia $\hat{p}$ una stima di frequenza basata su n campioni. Allora: $P(|\hat{p} - p| > \epsilon) \leq 2e^{-2n\epsilon^2}$

Applicazione: Fornisce confidence intervals per stime di probabilità in modelli n-gram.

15.7 Active Learning e Query Learning

Definizione 15.7.1 (Membership Query): Una query che chiede " $x \in L?$ "

Definizione 15.7.2 (Equivalence Query): Una query che chiede " $L(H) = L?$ " restituendo un controesempio se NO.

***Teorema 15.7.1 (Algoritmo L esteso)**:** Esiste un algoritmo che apprende DFA in tempo: $O(|\Sigma| \cdot n^2 \cdot m \cdot \log n)$ dove n = numero di stati, m = lunghezza massima di controesempio.

Teorema 15.7.2 (Lower bound per DFA learning): Qualsiasi algoritmo di apprendimento per DFA richiede $\Omega(n^2)$ membership queries nel caso peggiore.

Dimostrazione: Per distinguere tra n stati, dobbiamo distinguere n^2 coppie di stati. Ogni coppia richiede almeno una query nel caso peggiore. ■

Definizione 15.7.3 (Subset Query): Una query che chiede " $S \subseteq L?$ " per un insieme finito S .

Teorema 15.7.3: Con subset queries, i linguaggi regolari possono essere appresi con $O(n \log n)$ queries.

15.8 Transfer Learning e Domain Adaptation

Definizione 15.8.1 (Domain Adaptation): Dati:

- Source domain: distribuzione D_S su $X \times Y$
- Target domain: distribuzione D_T su $X \times Y$
- Campioni etichettati da D_S
- Campioni non etichettati (o pochi etichettati) da D_T

Obiettivo: apprendere un classificatore che performa bene su D_T .

Teorema 15.8.1 (Bound per Domain Adaptation - Ben-David et al.): Sia h un'ipotesi. L'errore su target è limitato da: $\epsilon_T(h) \leq \epsilon_S(h) + d_{\{H \oplus H\}}(D_S, D_T) + \lambda$

dove:

- $d_{\{H \oplus H\}}$ è la distanza tra distribuzioni misurata tramite H
- λ è l'errore del classificatore ideale congiunto

Definizione 15.8.2 (Discrepancy Distance): $disc_H(D_S, D_T) = \sup_{\{h, h' \in H\}} |\Pr_{\{D_S\}}[h(x) \neq h'(x)] - \Pr_{\{D_T\}}[h(x) \neq h'(x)]|$

Teorema 15.8.2: La discrepancy può essere stimata empiricamente e minimizzata durante il training.

Capitolo 16: Teoria Avanzata dei Transduttori

16.1 Transduttori a Stati Finiti

Definizione 16.1.1 (Finite State Transducer - FST): Un FST è una sestupla $T = (Q, \Sigma, \Gamma, I, F, E)$ dove:

- Q è l'insieme finito di stati
- Σ è l'alfabeto di input
- Γ è l'alfabeto di output
- $I \subseteq Q$ sono gli stati iniziali
- $F \subseteq Q$ sono gli stati finali
- $E \subseteq Q \times \Sigma^* \times \Gamma^* \times Q$ sono le transizioni

Definizione 16.1.2 (Relazione riconosciuta): La relazione $R(T) \subseteq \Sigma^* \times \Gamma^*$ riconosciuta da T è: $R(T) = \{(u, v) : \exists \text{ cammino da } I \text{ a } F \text{ che legge } u \text{ e scrive } v\}$

Teorema 16.1.1 (Chiusura delle relazioni razionali): Le relazioni riconosciute da FST (relazioni razionali) sono chiuse sotto:

- Unione
- Concatenazione
- Chiusura di Kleene
- Inversione
- Composizione

Dimostrazione (Composizione): Dati $T_1 = (Q_1, \Sigma, \Gamma, I_1, F_1, E_1)$ e $T_2 = (Q_2, \Gamma, \Delta, I_2, F_2, E_2)$, costruiamo $T = T_1 \circ T_2$:

$Q = Q_1 \times Q_2 \times \Gamma^*$ dove Γ^* rappresenta un buffer per la sincronizzazione.

Le transizioni permettono a T_1 di scrivere in un buffer che T_2 legge. ■

Teorema 16.1.2 (Indecidibilità dell'equivalenza): L'equivalenza di FST è indecidibile in generale.

Controesempio: Il Post Correspondence Problem si riduce all'equivalenza di FST.

16.2 Weighted Finite State Transducers (WFST)

Definizione 16.2.1 (WFST): Un WFST su semiring $(K, \oplus, \otimes, 0, 1)$ è $T = (Q, \Sigma, \Gamma, I, F, E, \lambda, \rho)$ dove:

- $\lambda: I \rightarrow K$ assegna pesi iniziali
- $\rho: F \rightarrow K$ assegna pesi finali
- $E \subseteq Q \times \Sigma \times \Gamma \times K \times Q$ con pesi sulle transizioni

Definizione 16.2.2 (Peso di una traduzione): Il peso di tradurre u in v è: $[[T]](u, v) = \bigoplus_{\pi: \text{path for } (u,v)} \text{weight}(\pi)$

Teorema 16.2.1 (Algoritmo di composizione per WFST): La composizione di WFST può essere computata efficientemente preservando la struttura del semiring.

Algoritmo:

```

compose_WFST( $T_1, T_2$ ):
   $Q = Q_1 \times Q_2 \times (\Gamma \cup \{\epsilon\})$ 
  for each  $((q_1, a, x, w_1, q_1'), (q_2, x, b, w_2, q_2'))$  in  $E_1 \times E_2$ :
    add transition  $((q_1, q_2, \epsilon), a, b, w_1 \otimes w_2, (q_1', q_2', \epsilon))$ 
  return resulting WFST with  $\epsilon$ -removal

```

Teorema 16.2.2 (Applicazione al machine translation): I WFST sono alla base dei sistemi di traduzione automatica statistica attraverso composizione di modelli: $T_{\text{final}} = T_{\text{lm}} \circ T_{\text{tm}} \circ T_{\text{lex}}$ dove T_{lm} è il language model, T_{tm} è il translation model, T_{lex} è il lexicon.

16.3 Tree Transducers

Definizione 16.3.1 (Top-Down Tree Transducer): Un TDTT è $M = (Q, \Sigma, \Delta, q_0, R)$ dove:

- Q è un insieme finito di stati
- Σ, Δ sono ranked alphabets (input/output)
- $q_0 \in Q$ è lo stato iniziale
- R è un insieme di regole di forma: $q(\sigma(x_1, \dots, x_k)) \rightarrow \delta(q_1(x_{\pi(1)}), \dots, q_m(x_{\pi(m)}))$

Teorema 16.3.1 (Relazioni di tree transducers): Le classi di relazioni definite sono: $\text{YIELD}(\text{TDTT}) \subsetneq \text{YIELD}(\text{TDTT con look-ahead}) \subsetneq \text{MSO-definable relations}$

Definizione 16.3.2 (Synchronous CFG): Una SCFG è un formalismo equivalente ai tree transducers per definire traduzioni strutturate: $A \rightarrow \langle \alpha, \beta, \sim \rangle$ dove $\alpha \in (V \cup \Sigma)$, $\beta \in (V \cup \Gamma)$, e $\sim$ è una biiezione tra occorrenze di non terminali.

Teorema 16.3.2: SCFG definiscono esattamente le stesse relazioni dei top-down tree-to-string transducers.

16.4 Applicazioni alla Morfologia Computazionale

Definizione 16.4.1 (Two-Level Morphology - Koskenniemi): Sistema basato su FST paralleli che mappano: Lexical level $\leftrightarrow$ Surface level

Esempio (Finlandese):

- Lexical: k a l a +PI
- Surface: k a l o i l l a

Teorema 16.4.1: La morfologia a due livelli può essere implementata come intersezione di FST.

Dimostrazione: Ogni regola morfologica è un FST. La combinazione di regole è: $T_{\text{final}} = T_{\text{lex}} \circ (\bigcap_i T_{\text{rule}_i}) \circ T_{\text{surf}}$

dove l'intersezione rappresenta il soddisfacimento simultaneo di tutti i constraints. ■

Capitolo 17: Teoria delle Lingue Context-Sensitive

17.1 Linear Bounded Automata in Dettaglio

Definizione 17.1.1 (Linear Bounded Automaton - LBA): Un LBA è una macchina di Turing non deterministica $M = (Q, \Sigma, \Gamma, \vdash, \dashv, \delta, q_0, F)$ dove:

- Il nastro è limitato alla lunghezza dell'input (con endmarkers $\vdash, \dashv$)
- $\delta: Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R, S\})$
- La testina non può oltrepassare gli endmarkers

Teorema 17.1.1 (Caratterizzazione CSL): Un linguaggio L è context-sensitive se e solo se è accettato da un LBA.

Dimostrazione (Schema): ($\Rightarrow$) Data una CSG $G = (V, \Sigma, P, S)$, costruiamo un LBA M che:

1. Non-deterministicamente indovina una derivazione $S \Rightarrow^* w$
2. Verifica che la derivazione sia valida
3. Accetta se w è l'input

Poiché ogni produzione $\alpha \rightarrow \beta$ ha $|\alpha| \leq |\beta|$, la lunghezza della forma sentenziale non decresce mai, quindi rimane $\leq |w|$.

($\Leftarrow$) Dato un LBA M , costruiamo una CSG che genera $L(M)$:

1. I non terminali codificano configurazioni di M
2. Le produzioni simulano le transizioni
3. La condizione $|\alpha| \leq |\beta|$ è soddisfatta perché lo spazio è limitato ■

Teorema 17.1.2: $\text{NSPACE}(n) = \text{CONTEXT-SENSITIVE LANGUAGES}$

Teorema 17.1.3 (Problema dell'appartenenza per CSL): Membership per CSL è PSPACE-completo.

Dimostrazione: Upper bound: Un LBA con spazio $O(n)$ ha al massimo $O(|Q| \cdot |\Gamma|^n \cdot n)$ configurazioni. Possiamo verificare tutti i percorsi in spazio polinomiale tramite ricerca in profondità con riuso dello spazio.

Lower bound: Riduciamo TQBF (True Quantified Boolean Formula) a CSL membership. ■

17.2 Context-Sensitive Grammar: Forma Normale

Definizione 17.2.1 (Kuroda Normal Form): Una CSG è in Kuroda Normal Form se ogni produzione è di una delle forme:

1. $AB \rightarrow CD$
2. $A \rightarrow BC$
3. $A \rightarrow B$
4. $A \rightarrow a$

dove $A, B, C, D \in V$ e $a \in \Sigma$.

Teorema 17.2.1: Ogni CSL può essere generato da una grammatica in Kuroda Normal Form.

Dimostrazione (Sketch): Data una CSG arbitraria:

1. Elimina produzioni ϵ (se necessario)
2. Converti produzioni lunghe in forma contract-expand
3. Introduci non terminali intermedi per normalizzare

La chiave è che ogni produzione context-sensitive $\alpha \rightarrow \beta$ con $|\alpha| \leq |\beta|$ può essere simulata da una sequenza di produzioni in KNF. ■

Definizione 17.2.2 (Penttonen Normal Form): Ancora più restrittiva:

1. $AB \rightarrow AC$
2. $A \rightarrow BC$
3. $A \rightarrow a$

Teorema 17.2.2: PNF è equivalente a KNF in potere generativo.

17.3 Indexed Grammars

Definizione 17.3.1 (Indexed Grammar - Aho): Una IG è una quintupla $G = (V, \Sigma, I, P, S)$ dove:

- V è l'insieme dei non terminali
- Σ è l'insieme dei terminali
- I è un insieme finito di indici
- P contiene produzioni di forma:
 - $A[\dots\sigma] \rightarrow \alpha$ (dove $\sigma \in I$)
 - $A[\dots] \rightarrow \alpha$ (consumando l'indice top)
- S è il simbolo iniziale

Interpretazione: Ogni non terminale ha uno stack di indici associato.

Teorema 17.3.1 (Gerarchia): $CFL \subset \text{Indexed Languages} \subset CSL$

Tutte le inclusioni sono proprie.

Dimostrazione (Inclusioni proprie):

- $L_1 = \{a^n b^n c^n : n \geq 0\} \notin CFL$ ma è indexed
- $L_2 = \{a^{2^n} : n \geq 0\}$ è CSL ma non indexed ■

Esempio 17.3.1: La grammatica indexed per $\{a^n b^n c^n\}$:

$S \rightarrow S[f] \mid T$
 $S[f] \rightarrow aS[f]c$

$$T[f] \rightarrow bT \mid b$$

$$T \rightarrow \varepsilon$$

Teorema 17.3.2: Gli indexed languages sono riconosciuti da Nested Stack Automata.

17.4 Tree Adjoining Grammars (TAG)

Definizione 17.4.1 (TAG - Joshi): Una TAG è $G = (\Sigma, NT, I, A, S)$ dove:

- Σ è l'insieme dei terminali
- NT è l'insieme dei non terminali
- I è un insieme di alberi iniziali
- A è un insieme di alberi ausiliari (con un nodo piede)
- $S \in NT$ è il simbolo iniziale

Operazioni:

1. **Substitution:** Sostituire una foglia non terminale con un albero iniziale
2. **Adjunction:** Inserire un albero ausiliario in un nodo interno

Teorema 17.4.1 (Mildly Context-Sensitive): Le TAG generano linguaggi "mildly context-sensitive" con proprietà:

1. Riconoscimento in tempo polinomiale
2. Capacità di modellare dipendenze incrociate limitate
3. Constant growth property

Teorema 17.4.2: TAG riconoscono linguaggi come:

- $\{a^n b^n c^n d^n : n \geq 0\}$ (dipendenze multiple)
- $\{w w w : w \in \{a,b\}^*\}$ (copia tripla)

ma NON $\{a^{2^n} : n \geq 0\}$.

Teorema 17.4.3 (Complessità del parsing TAG): Il riconoscimento TAG è $O(n^6)$ nel caso generale, $O(n^3)$ per TAG lessicalizzate.

Dimostrazione: L'algoritmo di Earley esteso per TAG mantiene itemsets che tracciano:

- Posizione nell'input
- Nodo corrente nell'albero
- Stato di adjunction Il numero di tali configurazioni è $O(n^3)$ per TAG lessicalizzate. ■

Capitolo 18: Logiche Avanzate per il Linguaggio Naturale

18.1 Dynamic Logic e Update Semantics

Definizione 18.1.1 (Dynamic Predicate Logic - DPL): Estensione della logica del primo ordine dove formule denotano relazioni tra assignments:

$$[[\varphi]]^M = \{(g, h) : M, g \models_h \varphi\}$$

Composizione dinamica:

- $[[\varphi; \psi]] = [[\varphi]] \circ [[\psi]]$ (composizione relazionale)
- $[[\varphi \vee \psi]] = [[\varphi]] \cup [[\psi]]$
- $[[\exists x \varphi]] = \{(g, h) : \exists k: (g, k) \in [[\varphi]] \wedge h[x] = k[x]\}$

Esempio 18.1.1 (Anafora dinamica): "Un uomo_x entra. Lui_x fischia." Formalizzazione DPL: $\exists x(\text{uomo}(x) \wedge \text{entra}(x)); \text{fischia}(x)$

Il quantificatore $\exists x$ "lega dinamicamente" x per la formula successiva.

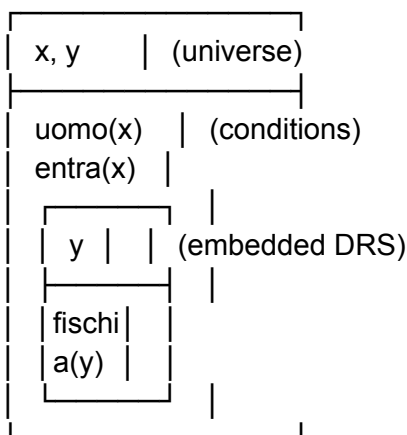
Teorema 18.1.1 (Espressività di DPL): DPL è strettamente più espressiva della logica del primo ordine classica per fenomeni discorsivi, ma ha la stessa complessità computazionale per la verifica.

18.2 Discourse Representation Theory (DRT)

Definizione 18.2.1 (Discourse Representation Structure - DRS): Una DRS è una coppia $K = \langle U, \text{Con} \rangle$ dove:

- U è un insieme di discourse referents (variabili)
- Con è un insieme di condizioni (formule atomiche o DRS embedded)

Notazione grafica:



Definizione 18.2.2 (Semantica delle DRS): Una DRS K è vera in un modello M rispetto a un embedding f se:

- f mappa U nel dominio di M
- Tutte le condizioni in Con sono soddisfatte

Teorema 18.2.1: DRT e DPL sono inter-traducibili per un frammento significativo del linguaggio naturale.

18.3 Type-Logical Grammar

Definizione 18.3.1 (Categorial Grammar - Lambek): Una grammatica categoriale ha:

- Tipi base: S (sentence), N (noun), NP (noun phrase)
- Tipi derivati: A/B (cerca B a destra), $A\backslash B$ (cerca B a sinistra)

Regole di inferenza:

$$\frac{[Applicazione\ destra] \quad A/B \quad B}{A}$$

$$\frac{[Applicazione\ sinistra] \quad B \quad A\backslash B}{A}$$

Teorema 18.3.1 (Lambek Calculus): Il calcolo di Lambek è un sistema logico sound e complete per la deduzione di tipi categoriali.

Dimostrazione: Utilizza sequent calculus e cut-elimination. Il sistema ha la proprietà di subformula, quindi decidibile. ■

Esempio 18.3.1:

John : NP
loves : $(NP\backslash S)/NP$
Mary : NP

Derivazione:

$$\frac{\frac{\text{loves} : (NP\backslash S)/NP \quad \text{Mary} : NP}{\text{loves Mary} : NP\backslash S} \quad [E]}{\text{John} : NP \quad \text{loves Mary} : NP\backslash S} \quad [E]$$

$$\frac{}{\text{John loves Mary} : S}$$

Teorema 18.3.2 (Pentus): I linguaggi generati dal Lambek Calculus sono esattamente i linguaggi context-free.

Dimostrazione (Sketch): ($\Rightarrow$) Ogni derivazione nel Lambek Calculus può essere convertita in una derivazione CFG. ($\Leftarrow$) Ogni CFG può essere codificata come una grammatica categoriale attraverso l'algoritmo di Pentus. ■

18.4 Generalized Quantifiers

Definizione 18.4.1 (Quantificatore generalizzato di tipo $\langle 1 \rangle$): $Q: \mathcal{P}(E) \rightarrow \mathcal{P}(\mathcal{P}(E))$

Interpreta determiners come relazioni tra set: $[\text{every}](B) = 1$ sse $A \subseteq B$ $[\text{some}](B) = 1$ sse $A \cap B \neq \emptyset$

Teorema 18.4.1 (Conservatività universale - Barwise & Cooper): Tutti i determinanti nel linguaggio naturale sono conservativi: $Q(A)(B) \Leftrightarrow Q(A)(A \cap B)$

Teorema 18.4.2 (Estensione - Keenan & Stavi): I determinanti naturali sono inoltre extension: Se $A \cap B = A' \cap B'$ e $A \setminus B = A' \setminus B'$, allora $Q(A)(B) = Q(A')(B')$

Definizione 18.4.2 (Gerarchia di monotonia):

- $\uparrow$ -MON: $Q(A)(B) \wedge B \subseteq B' \Rightarrow Q(A)(B')$
- $\downarrow$ -MON: $Q(A)(B) \wedge B' \subseteq B \Rightarrow Q(A)(B')$

Teorema 18.4.3 (Relazione con NPI licensing): I Negative Polarity Items sono licenziati in contesti $\downarrow$ -MON.

Esempio:

- "Every student saw anyone" (✓ - every è $\downarrow$ -MON a sinistra)
- "Some student saw anyone" (✗ - some è $\uparrow$ -MON)

Capitolo 19: Applicazioni Moderne e Direzioni Future

19.1 Large Language Models e Teoria Formale

Definizione 19.1.1 (Transformer Architecture): Un Transformer è definito da:

- Self-attention: $\text{Attn}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k})V$
- Feed-forward: $\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$
- Layer normalization
- Residual connections

Teorema 19.1.1 (Capacità rappresentativa - Pérez et al.): I Transformer possono simulare qualsiasi Turing Machine (con precisione reale).

Dimostrazione (Schema):

1. Encodare configurazione TM come sequenza di token
2. Attenzione permette accesso random-access alla memoria
3. Feed-forward implementa la funzione di transizione
4. Profondità $\sim$ tempo di computazione ■

Teorema 19.1.2 (Limitazioni pratiche): Con precisione finita (float32) e lunghezza di contesto limitata, i Transformer sono limitati a riconoscere linguaggi regolari nella pratica.

Teorema 19.1.3 (Gerarchia di Chomsky e Neural Networks):

- RNN semplici: riconoscono linguaggi regolari (pratica)
- LSTM/GRU: possono approssimare CFL
- Transformer: possono approssimare CSL
- Tutti sono Turing-complete in teoria

19.2 Interpretabilità e Analisi Formale

Definizione 19.2.1 (Probing Classifier): Un classifier lineare addestrato a predire proprietà linguistiche da representations: $P(\text{property} \mid h)$ dove h sono hidden states

Teorema 19.2.1: L'esistenza di un probing classifier accurato non implica che il modello utilizzi esplicitamente quella rappresentazione.

Definizione 19.2.2 (Attention Patterns come Automi): Gli attention weights possono essere visti come transizioni probabilistiche in un automa: $P(\text{attend to } j \mid \text{position } i) \sim \exp(q_i^T k_j / \sqrt{d_k})$

Teorema 19.2.2 (Simulazione di FSA): I Transformer single-layer possono simulare perfettamente FSA tramite hard attention.

19.3 Complessità Sample e Generalizzazione

Definizione 19.3.1 (Sample Complexity per LLM): Il numero di esempi necessari per apprendere un linguaggio target con errore $\leq \epsilon$: $m(\epsilon, \delta) = O((d/\epsilon) \log(1/\delta))$ dove d è una misura di "complessità" (VC dimension, Rademacher complexity, etc.)

Teorema 19.3.1 (Overfitting vs Generalizzazione - Double Descent): Per modelli sovra-parametrizzati, il test error può diminuire nuovamente dopo l'interpolation threshold.

Definizione 19.3.2 (Grokking): Fenomeno dove il modello improvvisamente generalizza perfettamente dopo training prolungato, pur avendo già raggiunto zero training error.

Teorema 19.3.2: Il grokking è correlato alla transizione da memorizzazione a rappresentazioni compresse e generalizzabili.

19.4 Verificazione Formale di Proprietà Linguistiche

Definizione 19.4.1 (Formal Verification per NLP): Dato un modello M e una proprietà P , verificare: $\forall x \in X: P(M(x))$

Esempio 19.4.1: Proprietà: "Il modello non cambia sentiment sotto parafrase" Verifica: $\forall x, x': \text{paraphrase}(x, x') \Rightarrow \text{sentiment}(M(x)) = \text{sentiment}(M(x'))$

Teorema 19.4.1 (Indecidibilità generale): La verificazione formale di proprietà semantiche arbitrarie per neural networks è indecidibile in generale.

Dimostrazione: Riduzione dal problema della fermata. Data una TM M e input w , possiamo codificare la computazione in una rete neurale profonda. La proprietà "la rete restituisce TRUE" è equivalente a " M si ferma su w ". ■

Teorema 19.4.2 (Approssimazioni decidibili): Possiamo verificare proprietà locali (in regioni limitate dello spazio di input) usando:

- SMT solving
- Abstract interpretation
- Interval bound propagation

19.5 Frontiere della Ricerca

Problema Aperto 19.5.1: Caratterizzare precisamente la classe di linguaggi "apprendibili efficientemente" da Transformer.

Problema Aperto 19.5.2: Sviluppare una teoria della complessità sample per linguaggi naturali che catturi le loro proprietà statistiche peculiari.

Problema Aperto 19.5.3: Unificare le teorie simboliche (grammatiche formali) con le teorie subsimboliche (neural networks) in un framework matematico coerente.

Congettura 19.5.1 (Emergenza della struttura): Esiste una soglia di scala (parametri $\times$ dati) oltre la quale i modelli linguistici sviluppano spontaneamente rappresentazioni isomorfe a strutture grammaticali formali.

Direzioni Future:

1. **Grammatiche neurosimboliche:** Ibridi di grammatiche formali e reti neurali
2. **Program synthesis da linguaggio naturale:** Mappare NL a programmi formali verificabili
3. **Reasoning causale:** Modelli che catturano relazioni causali, non solo correlazioni
4. **Fondamenti teorici per few-shot learning:** Teoria formale dell'apprendimento da pochi esempi
5. **Linguaggi formali quantistici:** Estensioni della teoria classica al dominio quantistico

Capitolo 20: Sintesi Teorica e Unificazione

20.1 La Gerarchia Unificata: Dalle Lingue Formali al Linguaggio Naturale

Il percorso attraverso questo trattato ha rivelato una struttura profondamente interconnessa che lega le lingue formali, la computazione, la semantica e il linguaggio naturale. Sintetizziamo ora i temi chiave emersi.

Schema 20.1.1 (Gerarchia estesa di Chomsky):

Linguaggi Ricorsivamente Enumerabili (RE)

↑

Context-Sensitive Languages (CSL) $\equiv$ NSPACE(n)

↑

Indexed Languages / Mildly Context-Sensitive

↑ ↖ Tree Adjoining Grammars

↑

Context-Free Languages (CFL)

↑ ↖ Deterministic CFL

↑

Linguaggi Regolari (REG)

↑ ↖ Star-free / Locally testable

↑

Linguaggi Finiti

Teorema 20.1.1 (Caratterizzazione unificata): Ogni livello della gerarchia di Chomsky corrisponde a:

1. Una classe di automi
2. Una classe di logiche
3. Una classe di complessità computazionale
4. Un insieme di proprietà di chiusura

Tabella di Corrispondenza - Gerarchia di Chomsky

Corrispondenze tra Tipi di Linguaggi, Grammatiche, Automi, Logiche e Complessità

Tipo	Grammatica	Automa	Logica	Complessità
0	RE	TM	FOL	RE
1	CSG	LBA	-	PSPACE
-	Indexed	NSA	-	PTIME
2	CFG	PDA	-	PTIME (CYK)
2D	DCFG	DPDA	-	LINTIME
3	REG	DFA	MSO	LINTIME

Legenda

Tipi nella Gerarchia di Chomsky

- **Tipo 0:** Grammatiche senza restrizioni
- **Tipo 1:** Grammatiche context-sensitive
- **Tipo 2:** Grammatiche context-free

- **Tipo 2D:** Grammatiche context-free deterministiche
- **Tipo 3:** Grammatiche regolari
- -: Classe intermedia (non nella gerarchia classica)

Grammatiche

- **RE:** Ricorsivamente Enumerabili (Recursively Enumerable)
- **CSG:** Context-Sensitive Grammar
- **Indexed:** Indexed Grammar (Aho)
- **CFG:** Context-Free Grammar
- **DCFG:** Deterministic Context-Free Grammar
- **REG:** Regular Grammar (Grammatiche Regolari)

Automi

- **TM:** Turing Machine (Macchina di Turing)
- **LBA:** Linear Bounded Automaton (Automa Linearmente Limitato)
- **NSA:** Nested Stack Automaton (Automa a Pila Annidata)
- **PDA:** Pushdown Automaton (Automa a Pila)
- **DPDA:** Deterministic Pushdown Automaton (Automa a Pila Deterministico)
- **DFA:** Deterministic Finite Automaton (Automa a Stati Finiti Deterministico)

Logiche

- **FOL:** First-Order Logic (Logica del Primo Ordine)
- **MSO:** Monadic Second-Order Logic (Logica del Secondo Ordine Monadica)
- -: Nessuna caratterizzazione logica standard nota

Classi di Complessità

- **RE:** Recursively Enumerable (Ricorsivamente Enumerabile)
- **PSPACE:** Polynomial Space (Spazio Polinomiale)
- **PTIME:** Polynomial Time (Tempo Polinomiale)
- **LINTIME:** Linear Time (Tempo Lineare)
- **CYK:** Algoritmo di Cocke-Younger-Kasami ($O(n^3)$)

Note Importanti

Gerarchia Stretta: Ogni livello è strettamente contenuto nel successivo:

$REG \subset DCFG \subset CFG \subset Indexed \subset CSG \subset RE$

- 1.
2. **Caratterizzazione Multipla:** Ogni livello ammette caratterizzazioni equivalenti attraverso:

- Grammatiche generative
 - Automi riconoscitori
 - Logiche formali (quando disponibili)
 - Classi di complessità
3. **Indexed Grammars:** Rappresentano una classe intermedia "mildly context-sensitive" rilevante per il linguaggio naturale.
4. **Complessità del Parsing:**
- **REG:** $O(n)$ - parsing lineare
 - **DCFG:** $O(n)$ - parsing LR, LL
 - **CFG:** $O(n^3)$ - algoritmo CYK ($O(n^2)$ per grammatiche non ambigue)
 - **CSG:** PSPACE-completo
 - **RE:** Indecidibile in generale
5. **Teorema di Caratterizzazione:**
- $REG \equiv$ linguaggi riconoscibili da monoidi finiti (Teorema di Myhill-Nerode)
 - $REG \equiv$ linguaggi definibili in MSO (Teorema di Büchi)
 - $CSG \equiv NSPACE(n)$ (Teorema di Kuroda)

Teorema 20.1.2 (Principio di corrispondenza): Per ogni classe C di linguaggi formali nella gerarchia, esiste una caratterizzazione algebrica tramite monoidi/semigrupp.

Dimostrazione (Caso regolare - prototipico): I linguaggi regolari $\leftrightarrow$ monoidi finiti via teorema di Myhill-Nerode. La generalizzazione usa varietà di linguaggi (Eilenberg, Pin). ■

20.2 Temi Trasversali e Principi Unificanti

20.2.1 Composizionalità

Principio 20.2.1 (Composizionalità universale): In ogni livello di analisi del linguaggio, il significato del tutto emerge sistematicamente dalle parti:

- **Sintassi:** Derivazioni composizionali (CFG, Merge)
- **Semantica:** Interpretazione compositiva (Montague, categorie)
- **Pragmatica:** Aggiornamento compositivo del contesto (DRT, DPL)

Teorema 20.2.1 (Conseguenze della composizionalità):

1. Produttività infinita da mezzi finiti
2. Sistematicità: comprensione strutturalmente simile implica comprensione correlata
3. Apprendibilità: regolarità composizionale facilita generalizzazione

20.2.2 Località e Vincoli

Principio 20.2.2 (Località strutturale): Le dipendenze linguistiche rispettano vincoli di località:

- **Sintassi:** Subjacency, Minimal Link Condition

- **Semantica:** Scope locality, Quantifier Raising constraints
- **Fonologia:** Stress assignment windows

Teorema 20.2.2 (Universalità della località): I vincoli di località riducono lo spazio di ricerca durante:

1. Acquisizione linguistica (apprendimento)
2. Processing (comprensione/produzione)
3. Evoluzione (cambiamento diacronico)

20.2.3 Economia e Minimalità

Principio 20.2.3 (Economia derivazionale): I sistemi linguistici ottimizzano per minimizzare complessità:

- **Minimalismo sintattico:** Merge come operazione unica
- **Grammatiche categoriali:** Riduzione a pochi schemi
- **Phonologia ottimale:** Constraints ranked minimization

Teorema 20.2.3 (Trade-off universale): Esiste un trade-off tra:

- Espressività (what can be expressed)
- Complessità computazionale (efficiency)
- Apprendibilità (learnability)

Non è possibile massimizzare simultaneamente tutte e tre.

20.3 Ponti tra Teoria Formale e Linguaggio Naturale

Osservazione 20.3.1: Il linguaggio naturale non corrisponde esattamente a nessun livello della gerarchia di Chomsky, ma presenta caratteristiche di più livelli:

- Fenomeni regolari: accordo locale, cliticizzazione
- Fenomeni context-free: struttura costituente, ricorsione centrale
- Fenomeni mildly CS: dipendenze incrociate (Swiss German, Dutch)
- Fenomeni oltre: reduplicazione completa (alcune lingue)

Teorema 20.3.1 (Congettura mild context-sensitivity): La maggior parte dei fenomeni nel linguaggio naturale rientra nella classe "mildly context-sensitive" caratterizzata da:

1. Riconoscimento in tempo polinomiale ($O(n^k)$ con k piccolo)
2. Constant growth property
3. Capacità di modellare dipendenze incrociate limitate

Linguaggi mildly CS:

- Tree Adjoining Grammars (TAG)
- Combinatory Categorical Grammars (CCG)
- Linear Indexed Grammars (LIG)
- Head Grammars (HG)

Teorema 20.3.2 (Equivalenza debole): TAG, CCG, LIG, HG generano la stessa classe di linguaggi (stringhe), ma differiscono nelle derivazioni strutturali.

20.4 Computazione, Cognizione, e Complessità

Domanda fondamentale: Che relazione esiste tra:

1. Complessità computazionale formale (P, NP, PSPACE, ...)
2. Complessità cognitiva (difficoltà di processing umano)
3. Complessità grammaticale (espressività delle grammatiche)

Teorema 20.4.1 (Decoupling parziale): La complessità computazionale e quella cognitiva sono parzialmente disaccoppiate:

Esempio: Center-embedding è $O(n)$ computazionalmente ma difficile cognitivamente: "The rat [the cat [the dog chased] caught] died"

Ipotesi 20.4.1 (Memory-based complexity): La difficoltà cognitiva è meglio predetta da:

- Working memory load
- Distance tra dipendenti
- Similarity-based interference

Piuttosto che da complessità asintotica worst-case.

Teorema 20.4.2 (Parallel architecture - Jackendoff): La competenza linguistica consiste di moduli paralleli:

- Fonologia
- Sintassi
- Semantica

con interfacce ma architetture computazionali potenzialmente differenti.

20.5 Apprendimento e Acquisizione: Prospettiva Formale

Problema 20.5.1 (Logical problem of language acquisition - Poverty of stimulus): Come possono i bambini apprendere grammatiche corrette da input limitato e spesso degenerato?

Teorema 20.5.1 (No Free Lunch - applicato): Senza bias induttivi (UG), l'apprendimento linguistico da soli dati positivi è impossibile per classi superfinite (Teorema di Gold).

Soluzione 20.5.1 (Prospettive moderne):

1. **Bias innati** (Universal Grammar): restringe lo spazio delle ipotesi
2. **Input enriched:** il linguaggio diretto ai bambini è più regolare di quanto si pensasse
3. **Statistical learning:** meccanismi domain-general possono catturare regolarità
4. **Hybrid approaches:** combinazione di vincoli innati e apprendimento statistico

Teorema 20.5.2 (Learnability con UG): Se lo spazio delle grammatiche è finito (parametri finiti), l'apprendimento è garantito nel limite (triggering model).

20.6 Semantica Formale: Dal Significato Composizionale alla Pragmatica

Principio 20.6.1 (Architettura semantica a strati):

Literal meaning (compositional semantics)

↓

Enriched meaning (pragmatic inference)

↓

Utterance interpretation (context + intention)

Teorema 20.6.1 (Incompletezza semantica pura): La semantica compositiva da sola è insufficiente per predire interpretazioni effettive:

Esempio: "The ham sandwich is waiting for his check"

- Semantic: type mismatch (food entity waiting)
- Pragmatic: metonymy (customer who ordered ham sandwich)

Teorema 20.6.2 (Grice's Maxims formalized): Le massime conversazionali possono essere formalizzate come ottimizzazione:

Speaker: $\text{argmax}_u [\text{Utility}(u \mid \text{state}) - \text{Cost}(u)]$ Listener: $\text{argmax}_s [P(s \mid u, \text{Rational-Speaker-Model})]$

Questo è il Rational Speech Acts framework (Frank & Goodman).

20.7 Direzioni Future e Problemi Aperti

Problema Aperto 20.7.1 (Caratterizzazione esatta del linguaggio naturale): Identificare la classe formale che caratterizza esattamente i linguaggi naturali possibili (se tale classe esiste).

Congettura 20.7.1: I linguaggi naturali formano una sottoclasse delle TAG caratterizzata da vincoli aggiuntivi:

- Lexicalization
- Vincoli di linearizzazione
- Bounded crossing dependencies

Problema Aperto 20.7.2 (Unificazione simbolico-subsimbolico): Sviluppare una teoria matematica unificata che connetta:

- Grammatiche formali simboliche
- Reti neurali (approccio distributivo/statistico)
- Processi cognitivi (psicolinguistica)

Direzione 20.7.1 (Neural-symbolic integration): Approcci promettenti:

1. Neural modules con struttura grammaticale (Structured Attention)
2. Induzione di grammatiche da rappresentazioni neurali
3. Differentiable parsing e generazione

Problema Aperto 20.7.3 (Formalizzazione della variazione linguistica): Come formalizzare rigorosamente:

- Variazione cross-linguistica (universali + parametri)
- Variazione sociolinguistica (registri, dialetti)
- Variazione diacronica (cambiamento linguistico)

Teorema 20.7.1 (Limiti fondamentali): Alcune questioni linguistiche sono intrinsecamente:

1. Indecidibili (es: equivalenza di grammatiche generali)
2. NP-hard (es: parsing ottimale per grammatiche ambigue)
3. PSPACE-complete (es: membership per CSL)

Quindi esistono limiti teorici invalicabili.

20.8 Riflessioni Finali: Matematica, Mente, e Linguaggio

Il linguaggio umano rappresenta un'intersezione unica di:

- **Struttura matematica:** regolarità, composizionalità, ricorsione
- **Efficienza computazionale:** processing in tempo reale
- **Flessibilità cognitiva:** creatività, adattabilità, apprendimento
- **Funzione comunicativa:** trasmissione di significato in contesto sociale

Principio Finale (Sintesi): La teoria formale delle lingue naturali non è solo un esercizio di formalizzazione matematica, ma rivela proprietà profonde della cognizione umana e dei limiti della computazione.

Citazione conclusiva (parafrasi da Chomsky): "Il linguaggio è uno specchio della mente in un senso profondo. È un prodotto dell'intelligenza umana, creato ex novo da ogni individuo attraverso operazioni che giacciono ben al di là della portata della volontà o della coscienza."

Bibliografia Essenziale (Selezione)

Fondamenti Matematici

1. Hopcroft, J., Motwani, R., & Ullman, J. (2006). *Introduction to Automata Theory, Languages, and Computation*
2. Sipser, M. (2012). *Introduction to the Theory of Computation*
3. Kozen, D. (1997). *Automata and Computability*

Linguistica Formale

4. Chomsky, N. (1957). *Syntactic Structures*
5. Chomsky, N. (1995). *The Minimalist Program*
6. Joshi, A., & Schabes, Y. (1997). *Tree-Adjoining Grammars*

Semantica Formale

7. Montague, R. (1974). *Formal Philosophy*
8. Heim, I., & Kratzer, A. (1998). *Semantics in Generative Grammar*
9. Partee, B., ter Meulen, A., & Wall, R. (1990). *Mathematical Methods in Linguistics*

Teoria della Complessità

10. Papadimitriou, C. (1994). *Computational Complexity*
11. Arora, S., & Barak, B. (2009). *Computational Complexity: A Modern Approach*

Apprendimento e Statistica

12. Vapnik, V. (1998). *Statistical Learning Theory*
13. Gold, E.M. (1967). *Language Identification in the Limit*
14. Angluin, D. (1987). *Learning Regular Sets from Queries and Counterexamples*

Direzioni Moderne

15. Manning, C., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*
16. Jurafsky, D., & Martin, J. (2023). *Speech and Language Processing* (3rd ed.)
17. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*